SBIR- 15.03-9030
release date 6/02/91 ✓

NASA-
NASA CR-
PSI-1038/TR-993

MULTICOLOR PYROMETER FOR MATERIALS PROCESSING IN SPACE

M.B. Frish, J. Frank, J.E. Baker, R.R. Foutter,
H. Beerman, and M.G. Allen
Physical Sciences Inc.
20 New England Business Center
Andover, MA 01810

PHASE II

Final Report and Operator's Manual

Items 0002 and 0004

Under Contract No. NAS7-1002

March 1990

Prepared for:

National Aeronautics and Space Administration
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

(NASA-CR-190876) MULTICOLOR PYROMETER FOR MATERIALS PROCESSING IN SPACE Final Report, 1 Jun. 1987 - 30 Nov. 1989 (Physical Sciences) 102 p

N93-16747

Unclas

~~H1~~ G3/29 0121196

National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Multicolor Pyrometer for Materials Processing in Space		5. Report Date
		6. Performing Organization Code
7. Author(s) M.B. Frish, J. Frank, J.E. Baker, R.R. Foutter, H. Beerman, and M.G. Allen		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address Physical Sciences Inc. 20 New England Business Center Andover, MA 01810		11. Contract or Grant No.
		13. Type of Report and Period Covered Phase II Final and Operators Manual - 6/1/87 - 11/30/89
12. Sponsoring Agency Name and Address		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report documents the work performed by Physical Sciences Inc. (PSI), under contract to NASA JPL, during a 2.5-year SBIR Phase II program. The program goals were to design, construct and program a prototype passive imaging pyrometer capable of measuring, as accurately as possible, and controlling the temperature distribution across the surface of a moving object suspended in space. These goals were achieved and the instrument was delivered to JPL in November 1989. The pyrometer utilizes an optical system which operates at short wavelengths compared to the peak of the black-body spectrum for the temperature range of interest, thus minimizing errors associated with a lack of knowledge about the heated sample's emissivity. To cover a temperatures from 900 to 2500 K, six wavelengths are available. The preferred wavelength for measurement of a particular temperature decreases as the temperature increases. Images at all six wavelengths are projected onto a single CCD camera concurrently. The camera and optical system have been calibrated to relate the measured intensity at each pixel to the temperature of the heated object. The output of the camera is digitized by a frame grabber installed in a personal computer, and analyzed automatically to yield temperature information. The data can be used in a feedback loop to alter the status of computer-activated switches and thereby control a heating system.

17. Key Words (Suggested by Author(s))	18. Distribution Statement

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	106	

NASA FORM 1626 OCT 86

CONTENTS

FIGURES

FIGURES

TABLES

SUMMARY

This report documents the work performed by Physical Sciences Inc. (PSI) under contract to NASA JPL, during a 2.5-year SBIR Phase II program. The program goals were to design, construct, and program a prototype passive imaging pyrometer capable of measuring, as accurately as possible, and controlling the temperature distribution across the surface of a moving object suspended in space. These goals were achieved and the fully-calibrated instrument was delivered to JPL in November 1989.

The multicolor imaging pyrometer was designed specifically to measure temperatures from about 900 to 2500 K. Its optical configuration was predicted on NASA's objective to measure and control the temperatures of new materials subjected to thermal processing in a spaceborne laboratory. An example of such a laboratory would be an acoustic leviation furnace, mounted abroad Space Shuttle or Space Station, in which material samples roughly 2 mm in diameter would be heated to temperatures as high as 2500 K by a combination of radiant heating from the furnace walls and laser irradiation. The materials are expected to change emissivity unpredictably as they heat and subsequently cool, either because of phase changes, chemical reactions, or other processes. In addition, the materials will move freely within a volume having a diameter roughly five times the sample diameter. Thus, the pyrometer was designed to enable observation of 2 to 3 mm diameter objects located anywhere within a 1 cm diameter volume.

Like all pyrometers, this instrument determines the temperature of a material by measuring the emitted radiation. However, unlike most other pyrometers, temperature measurement errors associated with a lack of knowledge about the heated sample's emissivity are minimized by utilizing an optical system which operates at short wavelengths compared to the peak of the blackbody spectrum for the temperature range of interest. In this regime the radiant power increases faster than exponentially with temperature. Because of this extreme sensitivity to temperature, the emissivity of the source plays a relatively small role in determining the emitted power. The short wavelengths therefore provide more accurate measurements of the temperature than can be made using longer wavelengths, assuming equally poor knowledge of the sample's emissivity. The penalty paid for this accuracy is that the dynamic range of the pyrometer's CCD image detector places rather narrow limits on the range of temperatures that can be measured at a single short wavelength. To cover a broad range of temperature, six wavelengths - each sensitive to a specific temperature range - are provided.

An optical head projects distinct images at all six wavelengths onto the detector concurrently. Computerized data acquisition and analysis enables the user to capture data and store it on videotape, interactively examine and interpret the data either immediately after capture or upon replay of the videotape, or to automatically analyze the pyrometer's output. In the latter mode, the pyrometer provides a display of the object's temperature distribution in a false color image on a video monitor, a calculation of the temperature distribution's mode value, and a capability to use that value in a feedback loop for controlling the object's temperature. To enable user modifications, descriptions of the software have been supplied.

INTRODUCTION

Pyrometry is a well-established technique for determining the temperature of a material by measuring the radiation that it emits. In general, the accuracy of pyrometry suffers from the problem that there is always one parameter which must be known to determine the temperature but cannot be measured.[1-5] This parameter is usually the emissivity of the material at a specific wavelength or the ratio of emissivities at two or more different wavelengths. In most materials the emissivity changes as the material is heated, and easily-made measurements of the emissivity at room temperature provide little information that is useful at elevated temperatures. Therefore, an educated guess of the material's high temperature emissivity must be made. The accuracy of this guess determines the resultant accuracy of the temperature measurement.

The PSI multicolor imaging pyrometer was designed specifically to measure temperatures from 1000 to 2500 K with a minimum of emissivity related uncertainty. Its optical configuration was predicated on NASA's objective to measure and control the temperatures of new materials subjected to thermal processing in a spaceborne laboratory. An example of such a laboratory would be an acoustic leviation furnace, mounted aboard Space Shuttle or Space Station, in which material samples roughly 2 mm in diameter would be heated to temperatures as high as 2500 K by a combination of radiant heating from the furnace walls and laser irradiation. The materials are expected to change emissivity unpredictably as they heat and subsequently cool, either because of phase changes, chemical reactions, or other processes. In addition, the materials will move freely within a volume having a diameter roughly five times the sample diameter. Thus, the pyrometer was designed to enable observation of 2 to 3 mm diameter objects located anywhere within a 1 cm diameter volume. In addition, the pyrometer has the capability to electronically vary the image exposure time, thereby enabling freezing of rapidly moving objects at the expense of low temperature capability or of improving its low temperature capability (compared to nominal design values) by increasing the exposure time.

These features are realized through an optical system that collects radiation emitted at wavelengths that are short compared to the peak of the blackbody spectrum for the temperature range of interest. As discussed in Appendix A, in this regime the radiant power increases faster than exponentially with temperature. Because of this extreme sensitivity to temperature, the emissivity of the source plays a relatively small role in determining the emitted power. The short-wavelengths therefore provide more accurate measurements of the temperature than can be made using longer wavelengths, assuming equally poor knowledge of the sample's emissivity. However, the dynamic range of the pyrometer's variable-integration-time CCD image detector (manufactured by Sierra Scientific and described in Appendix B) places rather narrow limits on the range of temperatures that can be measured at a single short wavelength. To cover a broad range of temperatures, six wavelengths - each sensitive to a specific temperature range while providing both high accuracy and precision - are provided. An optical head, details of which comprise Appendix C, has been constructed to project distinct images at all six wavelengths onto the detector concurrently. The pyrometer is provided with electronics to control the detector (described in Appendix D) and a computerized frame acquisition, digitization and analysis system to facilitate properly-timed capture of the images regardless of exposure time. It has been calibrated at PSI using the procedure reported in Appendix E. Software, fully described in Appendix F, is supplied to enable the user to capture data and store it on videotape, interactively examine and interpret the data either immediately after capture or upon replay of the videotape, or to automatically analyze the pyrometer's output resulting in a display of the object's temperature distribution in a false color image on a video monitor, a calculation of the temperature distribution's mode value, and a capability to use that value in a feedback loop for controlling the object's temperature.

SYSTEM ARCHITECTURE

The heart of the pyrometer is the computer-coupled data acquisition and control system, constructed around an ALR 25386DT personal computer. The core of the image analysis system is the Data Translation Model DT-2861 frame-grabber. This device is entirely contained on a board that plugs into one of the computer slots. It incorporates an 8-bit A/D flash converter that digitizes the incoming video information at 10 MHz, converting the analog signal into a stream of digital values or "gray levels" between 0 and 255.

The sequence of events yielding one frame of data is as follows: The camera's CCD detector array is exposed to light, i.e., a flux of photons, from the optical head. The photons generate photoelectrons at the 610 x 245 individual CCD photosensitive sites with a probability equal to the quantum efficiency of the detector at the photon wavelength. Electrical charge is collected for a period of time determined by the computer-controlled interface described in Appendix E. At the end of this integration period, the accumulated charge is tranferred out of the CCD array, passed through a trans-impedence amplifier, and converted to an output voltage. The charges from the pixels are read out sequentially, resulting in a temporally varying voltage signal, the voltage at any given time being directly proportional to the charge accumulated at the corresponding pixel site during the integration time. The camera circuitry formats the output into the RS-170 composite video standard[6] (see Appendix B), and this signal is coupled into the frame grabber and digitized. The frame-grabbing function is synchronized by the timing information contained in the RS-170 composite video output of the camera, and is asynchronous with respect to the camera pixel clock. The results of the digitization are stored in a 512 x 512 block of computer memory, called a frame buffer, on the frame-grabber board. The data stored in the frame buffer can be displayed on a video monitor (or written to a mass storage device) in the form of 512 horizontal video lines, each segmented into 512 columns.

The DT-2861 frame-grabber was selected because of its 16-frame storage capacity and on-board Arithmetic Logic Unit (ALU). These two features allow storage and real-time subtraction of a background frame, a function that is required for accurate radiometric temperature measurement. In addition, the DT-2861 can be triggered externally, enabling continuous display of data while awaiting input of the discontinuous data supplied by the Sierra Scientific camera when operated in the variable integration time mode. In a typical temperature measurement, as the analog video data provided to the frame grabber from the camera is digitized, the ALU subtracts the background frame pixel-by-pixel and stores the resultant image in a frame buffer. Concurrently, the information in the storage buffer is passed out to a display device (i.e., a monitor) through an output lookup table. This table relates each of the 256 gray scales to three distinct 8-bit parameters corresponding to the intensity to be displayed by each of the red, green, and blue drives of the video display monitor. Thus, each of the RGB drives has 256 possible levels, providing a display palette of 2^{24} colors. This false color encoding is used by the example program PYRO to map the temperature information encoded in the gray scales to a set of color scales.

On-board frames in the DT-2861 can be transferred over the computer bus to system memory for analysis. This feature enables use of the video data for temperature control applications. To this end, the data acquisition and control system has also been configured with a Data Translation Model DT-2801A plug-in board. This device provides 16 channels of 12-bit A/D input, 2 channels of 12-bit D/A output, and 2 bytes of digital input or output. One of the D/A outputs is used by the camera interface to automatically control the exposure time. The digital outputs provide convenient process control switches.

The software environment driving the system is constructed from a combination of subroutine packages provided in conjunction with the hardware, and PSI-written algorithms which use the subroutines. The frame-grabber is controlled by a set of callable subroutines in a package provided by Data Translation called DT-IRIS. They operate with FORTRAN, PASCAL, or C languages. A similar package called PCLAB provides subroutines to drive the DT-2801A board. In addition, a package called IRIStutor provides an interactive learning environment which is useful for demonstrating the features of the frame grabber. All of the software runs within the MS-DOS operating system.

Full manufacturer's documentation of the hardware and software packages has been provided with the pyrometer. This includes operator's manuals for the the DT-2861 frame-grabber and the DT-2801A data acquistion board, and user's manuals for DT-IRIS and PCLAB. IRIStutor is described in the DT2861 manual. PSI recommends that the user become familiar with this documentation prior to operating the pyrometer. Understanding the system operation will ultimately facilitate user modifications of the supplied programs, thereby maximizing the usefulness of the pyrometer.

PYROMETER COMPONENTS AND INITIAL SETUP

The multicolor imaging pyrometer has three major components: 1) The optical head which collects radiation from the heated object, images the radiation from six selected narrow bandpasses onto the CCD detector, and converts the output of the detector to a standard video signal; 2) An interface box which provides the signals needed to operate the CCD detector camera in both its standard video and its variable-integration-time modes, synchronizes and couples the output of the camara with the data acquisition system, and couples the outputs of the data analysis hardware with a video monitor and with user-supplied temperature control equipment; and 3) The data acquisition and analysis system which is comprised of the computer and monitor, frame-grabber and RGB video monitor, appropriate software, and the DT-2801A control board. These components have been assembled, aligned, calibrated and tested at PSI before shipment to NASA. However, the optics may have become misaligned during shipment, and should be aligned before operation. The alignment procedure is decribed later.

Setting up the pyrometer should be a simple matter of unpacking the components and plugging-in the appropriate cables to connect them, as follows:

1. Connect the optical head to the interface box using the cable supplied. The twelve-pin circular connector plugs directly into the rear of the Sierra Scientific camera attached to the optical head. The 15-pin connector attaches to the front of the interface box.

2. Connect the 16-pin 3M-type connector to the upper frame-grabber port on the back of the computer. Connect short BNC-cables to the six labeled BNC connections at the opposite end of this cable. Plug opposite ends of the six BNC cables to the appropriately labeled connections on the back of the interface box.

3. Connect one end of the long BNC cable to the "Computer Sync" output on the front of the interface box, and the other end to the wire-wrap connections to the frame-grabber on the back of the computer.

4. Connect the 24-pin video monitor cable from the back of the interface box to the back of the video monitor.

5. Connect the screw terminal panel to the multipin terminal labelled "D/A Strip" on the front of the interface box. Connect the colored ribbon cable from the D/A terminal on the back of the interface box to the DT-2801A port on the back of the computer.

6. Plug the computer monitor and keyboard into the back of the computer. Release the hard disk drive from its parked position.

7. Plug the computer, interface box, and video monitor into 120V outlet.

To test the system, turn on the computer, video monitor, and interface box. Select "Standard Video" on the front of the interface box. Place a small

(~ 2 mm diameter) illuminated object (such as a light bulb positioned behind an aperture) 36 cm in front of the pyrometer's objective lens. After the computer boots, its monitor should read "C: \MCIP >". Type "cd \iris" (without the quotes) RETURN. The computer should read "C: IRIS >". Type "ir" RETURN. When the IRIS-tutor program begins, type "sync external" RETURN and "display camera" RETURN. Several images of the object should appear on the video monitor. Adjust the position of the object slightly to focus the images.

To test the variable integration time mode, select "Variable" on the front of the interface box, along with "Internal Control". Adjust the timer knob to a value less than 400. Press the "Reset" button and hold for at least 1 second. Images, brighter than those seen using standard video, should now flash on the monitor. To stop the flashing, exit IRIStutor by pressing key F10, followed by "Y" RETURN. Then type "cd \mcip" RETURN. The computer will respond with "C: \MCIP >". Type "test" RETURN and a program will begin running that uses the external trigger circuit to update the video display only when a new data-bearing image is acquired. Should no image appear, press the "Reset" button again for one second or more. Press any key to terminate the program.

ALIGNMENT AND GAIN ADJUSTMENT

For the pyrometer to provide correct temperature information, the optics must be properly aligned, and the camera's video gain adjusted to correspond to that at which it was calibrated. These adjustments are most easily made with the aid of a small He-Ne laser, a variable brightness white light such as an incandescent bulb, a variable aperture, and a blackbody source capable of achieving 1000 C. To align, the following procedure should be followed after reading Appendix C and noting that steps 2 - 4 should only be needed in the case of gross misalignment, resulting from displacement of the epoxied optical mounts:

1. Remove the outer side cover from the optical head.

2. Direct the He-Ne laser beam into the pyrometer through the center of the objective lens. Adjust the beam so that is parallel to the centerline of the upper row of dichroic beamsplitters.

3. Adjust the first beamsplitter so that the laser beam is approximately centered on the mirror that directs the beam into the lower row of dichroics. Then adjust the mirror so that the beam reflected from it is parallel to the centerline of that row. Slight readjustments of both the beamsplitter and mirror may be necessary.

4. Adjust the dichroic beamsplitters and mirrors in both the upper and lower rows so that the portion of the laser beam reflected from each one is approximately centered in the corresponding interference filter.

5. Remove the laser and position the light source behind a 2 to 2.5 mm diameter aperture centered along the optical axis 36 cm from the flat front surface of the pyrometer (not including the lens surface or lens mounting flange).

6. Run the IRIStutor program as described in the initial set-up procedure. Use the PLACE command to place a cursor at coordinates (128,320) on the video monitor. Mark that position with a removable sticker or grease pencil. Then use "display camera" to monitor the pyrometer's video output.

7. Increase the brightness of the light so that the long-wavelength images are visible on the monitor but not saturated. By placing a hand between the camera and the imaging lenses, turn the upper lens closest to the objective lens (which corresponds to the longest wavelength) to bring the aperture into best focus. Then turn the corresponding dichroic beam-splitter mount adjustment screws to center the image on the mark at coordinates (128,320).

8. Repeat the process for all other wavelengths, increasing the brightness of the bulb as required to make each image detectable and centering the images at the coordinates given in Appendix C, Eq. (C.1). The integration

time may need to be increased to visualize the shortest wavelength images. Use the variable-integration-time mode and the "test" program as required. Do not become concerned about blurring of the long wavelength images when they are oversaturated. This is primarily an effect of internal reflections and will not damage the camera.

GAIN CALIBRATION

The Sierra Scientific camera has a user-adjustable video gain, accessible through a small hole in the side of the camera housing. This gain has been adjusted at PSI so that the camera's RMS noise is approximately one gray level when the camera output is 255 gray levels. This gain setting may have changed during shipment. It is best checked by using the pyrometer to measure the temperature of a blackbody source at a known temperature. To this end, configure the blackbody source with a 2 to 2.5 mm aperture positioned 36 cm from the pyrometer and centered on the optical axis as described above. Allow the blackbody to equilibrate at a known temperature, preferably at or above 900 C. Run the PYRO program, entering an emissivity of 1.0 when requested. The computer monitor should display the correct temperature. If not, the camera gain should be varied slowly until the correct temperature is shown.

The camera also has a user-selectable offset, which controls the camera's output voltage in the absence of any light input, called the "black level". The actual value of this offset is not critical, since all measurements made with the pyrometer first subtract a background frame containing the black levels of all pixels. However, it is important that the offset be positive. Should the offset become negative, then the frame grabber will interpret it to be zero and thus perform the background subtraction incorrectly. A negative offset is indicated when the area not illuminated by the six pyrometer images becomes totally black, whereas during normal operation with background subtraction this area has gray levels between 0 and 2. To correct a negative offset, the Sierra Scientific camera must be removed from the pyrometer and its cover opened as described in the camera manual. The offset screw inside the camera should be adjusted to provide average black levels of about 5 digitizer steps, and the camera then reassembled and installed in the pyrometer. Be sure to check its vertical alignment.

USING THE PYROMETER

As described above, the pyrometer projects six non-overlapping images of a heated object onto the CCD array simultaneously, but for each point on the object only one or two images provide data suitable for reduction to temperature. The corresponding point in each of the other images is either bright enough to saturate the frame grabber, or dim enough to be undetectable. Four steps are required for each update of the temperature map acquired and displayed with the pyrometer: 1) acquiring a field of data; 2) subtracting the dark signal from that field; 3) selecting the appropriate one of the six available images for analysis; and 4) converting the radiance data in the image to temperature values.

The pyrometer has been supplied with three computer programs that are designed to aid the user in interactively following these four steps and thereby obtaining the maximum amount of information from the pyrometer, or to use the instrument as an automatic temperature controller. The programs QUICKIE and MANUAL allow the user to acquire and store (on videotape for example) unanalyzed data as rapidly as the CCD camera can provide it, i.e., 60 Hz in standard mode, slower in variable integration time mode. The data can be played back frame-by-frame at the users convenience and analyzed pixel-by-pixel. Dark signal subtraction can be performed during data acquisition or upon playback. Alternatively, the program PYRO automatically and repetitively performs all four steps and can use the results to control the heating of the object. For each frame of data collected, PYRO analyzes the six images to determine the most prevalent object temperature and presents that information on the monitor. The measured temperature is also supplied it to an externally-callable subroutine that can turn switches on or off, thus providing the control function.

Details of these programs are provided in Appendix F. Suggested modes of operation are discussed below.

Interactive Data Collection and Analysis

To collect data that can be subsequently analyzed interactively, use of the QUICKIE program is recommended. QUICKIE simply acquires video frames from the pyrometer, in standard or variable integration time modes, and displays them, in black and white, on the video monitor. Background subtraction of the dark signal at the selected integration time can be performed if so desired. The average dark signal of ten frames is acquired before collecting data by covering the pyrometer's aperture when prompted by the computer. This average dark frame is then subtracted from each frame of data before displaying the data on the monitor. It is recommended that this feature be utilized to simplify subsequent analysis. The data can be stored on videotape simply by connecting any one of the three RGB outputs from the frame grabber to the input of the videotape recorder. Use of a professional-quality or Super-VHS recorder is recommended to ensure accurate reproduction of the data upon playback.

The data collected using OUICKIE may be analyzed, frame-by-frame, with the MANUAL program. As described in Appendix F, MANUAL provides for interactive measurement of the temperature at each pixel in the image. The recommended procedure is to disconnect the pyrometer's video output cable on the rear of the electronics box from the frame grabber, and then connect the output of the videotape recorder to the frame grabber input. Then run MANUAL, using standard video rate and without background subtraction, while playing-back the videotape. Pause the videotape when a frame suitable for analysis is reached, and press the HOME key as prompted. The images acquired are then processed to display the most prevalent temperature, as with the PYRO program discussed below, and false coloring is provided. Upon completion of the analysis, the user is given the option for interactive measurement of temperatures at each pixel. The pixels are selected by using the arrow keys to move a cursor along the video monitor screen. After analysis is completed, a new frame can be selected by pressing the ENTER key, advancing the videotape, and starting again. Note that MANUAL can be used to analyze data provided directly by the pyrometer, rather than from videotape, if so desired. This capability would be useful to recalibrate the instrument, for example.

Automatic Data Collection and Process Control

The PYRO program, as delivered, collects data, processes it to determine object temperature distributions, and presents the results to the user. The temperature measurement procedure combines all of the steps described above. Upon program startup, six lookup tables converting the gray levels in each of the six images into temeperature values are calculated after the user inputs his or her best estimate of the object's emissivity. Each temperature value is assigned a color for display on the monitor. The lookup tables are stored in appropriate buffers on board the frame grabber. During operation of PYRO, the following steps are followed repetitively: A video field containing the six images of the hot moving object is acquired and, after background subtraction, stored in a frame buffer. The image containing the most useful data is then located by scanning predetermined portions of the video field to locate pixels which have non-zero but unsaturated gray levels. The lookup table corresponding to that image is selected and the images are displayed on the video monitor with appropriate false coloring. The selected image is indicated on the computer monitor.

The program also determines the most prevalent temperature within the selected image and provides that value to a subroutine called CONTROL. As described in Appendix F, CONTROL is meant to be a user-written subroutine which can be linked with PYRO.OBJ. The CONTROL subroutine can be used in conjunction with the DT-2801A board to alter the status of the digital output ports and, by connecting the digital outputs to suitable switches or relays, control the heating process. The subroutine can, for example, be a simple algorithm which turns a switch on or off depending on whether a preset control temperature has been reached, or it can be a program that simulates a proportional temperature controller, or any other routine that suits the needs of the user.

REFERENCES

1. M.B. Frish, "Measurement of High Temperature Thermophysical Properties with Electron Beam Heating," AIAA Paper No. 84-1784 (1984).

2. M.B. Frish, M.N. Spencer, N.E. Wolk, J.S. Werner and H.A. Miranda, "Multi-Color Pyrometer for Materials Processing in Space", Proc. of the Non-contact Temperature Measurement Workshop, NASA Conf. Pub. 2503 (May 1987).

3. P.C. Nordine, "The Accuracy of Multicolor Optical Pyrometry", High Temperature Science, 21, (1986).

4. D.P. DeWitt and R.E. Rondeau, "Evaluation of a Method for Measuring Front-Face Surface Temperatures and Spectral Emissivities During Irradiation", AIAA Paper No. 87-1565 (1987).

5. Siegel, R. and Howell, J.R., Thermal Radiation Heat Transfer, McGraw-Hill (1981).

6. B. Grob, Basic Television Principles and Servicing, McGraw-Hill (1975).

APPENDIX A

The Accuracy of Multicolor Pyrometry

The radiant energy emitted by any object is determined by the Planck equation:

$$R(\lambda)d\lambda = \frac{C_1}{\pi\lambda^5} \frac{\varepsilon(\lambda,T)\, d\lambda}{\exp(C_2/\lambda T) - 1} \tag{A.1}$$

where $R(\lambda)d\lambda$ is the radiant flux per steradian per unit area of emitter surface in the wavelength interval $[\lambda, \lambda + d\lambda]$, $C_1/\pi = 1.191 \times 10^{-12}$ W-cm^2/sr, $C_2 = 1.44$ cm-K, T is the temperature, and $\varepsilon(\lambda,T)$ is the spectral emissivity of the material at wavelength λ. In a typical pyrometer, a portion of the heated target's surface is imaged by a group of lenses and mirrors onto a photodetector which converts the incident radiation into a measurable electrical quantity. The optical path generally contains several windows, mirrors, beam splitters or filters, each of which has a wavelength dependent transmittance or reflectance, $t_i(\lambda)$. They, together with the solid angle Ω subtended by the optical collection system, and the surface area, A_s, of the radiant target, determine the radiant power which reaches the detector,

$$P = \frac{A_s \Omega C_1}{\pi} \int_0^\infty \frac{\prod_{i=1}^{n} t_i(\lambda)\varepsilon(\lambda,T)d\lambda}{\lambda^5[\exp(C_2/\lambda T) - 1]} \tag{A.2}$$

where n is the number of components in the optical train. By using a narrow bandpass optical filter to select a particular wavelength (i.e., color) at which to measure the radiance, the emissivity of the target and the transmittance or reflectance of other optical components are essentially constant over the bandpass. These variables can therefore be removed from the integral in Eq. (A.2) and, because the transmission curve of the filter is known, the integral may be evaluated, as

$$P = \varepsilon_\lambda D_\lambda \int_{\lambda_1}^{\lambda_2} \frac{t_f(\lambda)d\lambda}{\lambda^5[\exp(C_2/\lambda T) - 1]} \tag{A.3}$$

where ε_λ is the emissivity at the central wavelength of the filter, D_λ is a constant (independent of temperature) determined by the optical system and which may be evaluated by calibration, t_f is the (known) transmission function of the filter, and λ_1, λ_2 are the bandpass limits of the filter. When the photodetector is operated such that it generates a voltage signal proportional to the energy incident on its surface during a period of time τ, Eq. (A.3) may be written as

PRECED[illegible] NOT FILMED

$$V(T) = \varepsilon_\lambda B_\lambda F_\lambda(T) \tag{A.4}$$

where $F_\lambda(T)$ is a known thermal response function proportional to the integral in Eq. (A.3), and

$$B_\lambda = GD_\lambda \eta_\lambda \tau \tag{A.5}$$

where G (volts/coulomb) is the responsivity of the photodetector and its associated amplification circuitry, and η_λ (coulombs/joule) is proportional to the quantum efficiency of the detector.

Unfortunately, in addition to its dependence on temperature, the radiant emission from a heated object at a given wavelength depends on its emissivity, an intrinsic property of the material. Furthermore, as indicated in Eq. (A.1), the emissivity may be a function of both wavelength and temperature. It may also change with time as a material suffers changes in its transparency, reflectivity, or surface structure due to phase changes, chemical reaction, ablation, etc. Thus, the wavelength and temperature dependences of the emissivity are often unknown, and precise radiometric measurement of the true temperature is such situations is generally accepted as being essentially impossible.

By always operating at wavelengths such that $\lambda T << 1$ cm-K, the multicolor technique employed by the PSI pyrometer provides the best possible estimate of the true temperature under the difficult circumstances of unknown and unmeasureable emissivities which vary wildly and unpredictably with wavelength, temperature, and time. Eq. (A.1) shows that, at these short wavelengths, the radiant power emitted by a heated surface increases faster than exponentially with temperature, but is only linearly dependent on emissivity. Thus, a relatively large uncertainty in emissivity causes only a small error in temperature. Mathematically, this is seen by solving Eq. (A.4) for temperature. The function $F_\lambda(T)$ may be easily evaluated by approximating the transmission function of the narrow bandpass filter by a rectangle of height t_λ and width $\Delta\lambda = \lambda_2 - \lambda_1$. Defining

$$F_\lambda(T) = 1/[\lambda^5(\exp(C_2/\lambda T) - 1)] \tag{A.6}$$

and

$$B_\lambda = \frac{A_s \Omega t_\lambda \eta_\lambda \tau G C_1 \Delta\lambda}{\pi} \prod_{i=1}^{n} t_i \tag{A.7}$$

and solving Eqs. (A.4) through (A.7) for temperature yields

$$T = \frac{C_2}{\lambda \ln([B_\lambda \varepsilon_\lambda / V] + 1)} \quad . \tag{A.8}$$

Differentiating with respect to ε_λ gives the temperature accuracy as

$$\frac{\Delta T}{T} = [1 - \exp(-C_2/\lambda T)]\,\frac{\lambda T}{C_2}\,\frac{\Delta\varepsilon_\lambda}{\varepsilon_\lambda} \tag{A.9}$$

which is plotted in Figure A.1 for several wavelengths using $\Delta\varepsilon_\lambda/\varepsilon_\lambda \approx \pm 0.25$.

On the basis of Eqs. (A.8) and (A.9), it would appear that a single color pyrometer could be used to measure any temperature to any degree of accuracy simply by selecting a sufficiently short wavelength. Although this is true in principle, either detector sensitivity or shot noise places a lower limit on

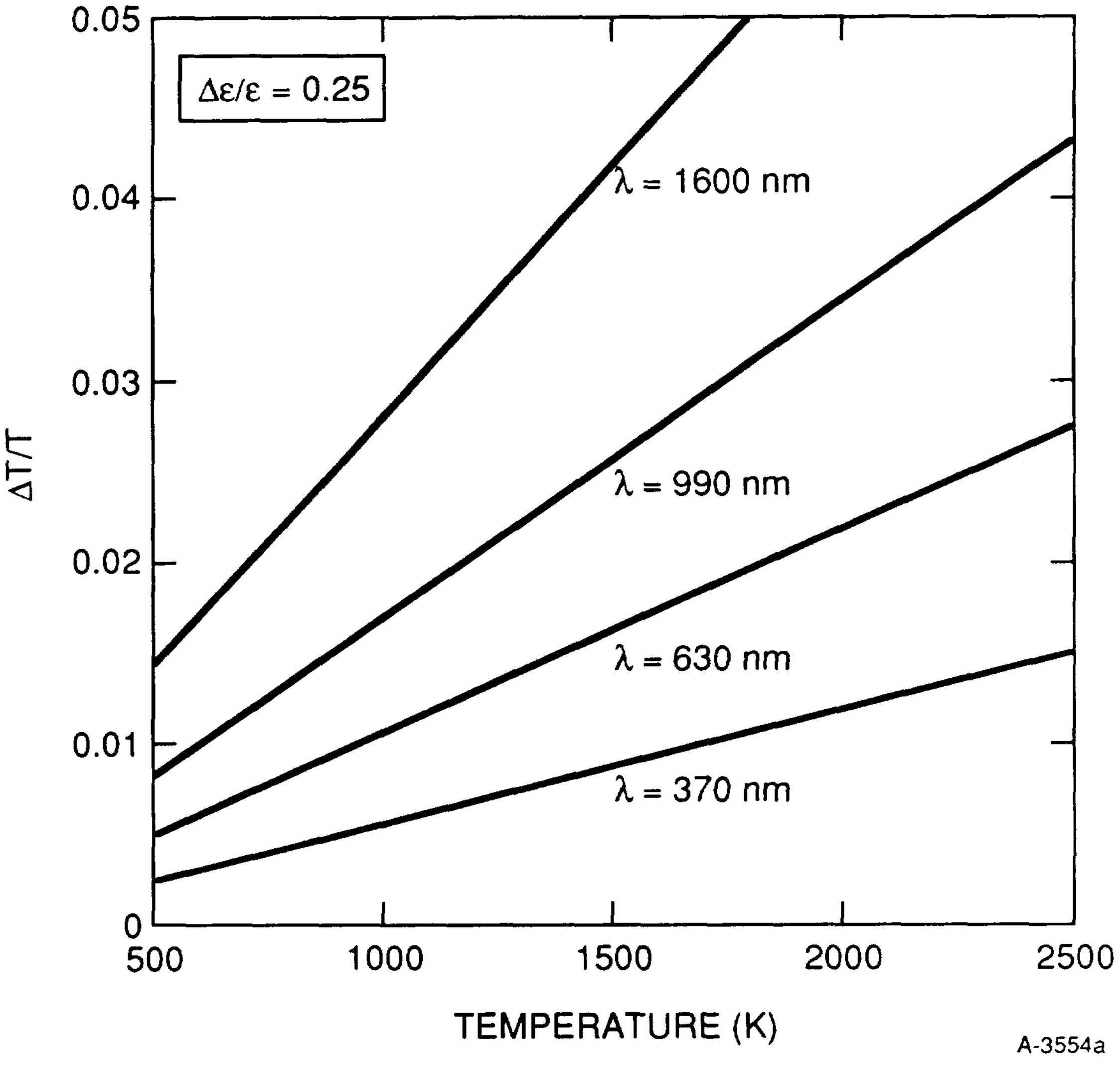


Figure A.1. - Temperature errors resulting from $\pm 25\%$ emissivity error at four wavelengths.

the temperature sensitivity for any particular wavelength and optical collector combination. In addition, there is a maximum temperature to which a particular system will be sensitive, fixed by the onset of detector saturation. Photodetectors used in typical imaging systems have a dynamic range of only about two orders of magnitude. However, at 370 nm, where the temperature accuracy is ± 40 K when the true temperature is 2500 K and the emissivity error is ± 25 percent, the radiant power spans a dynamic range exceeding ten orders of magnitude as the temperature increases from 1000 to 2500 K. A single imaging pyrometer operating at this wavelength is clearly unsuitable for measurements over this entire temperature range. Longer wavelengths cannot be used at the higher temperatures without sacrificing accuracy. However, if use of this short wavelength is limited to temperatures between about 2250 and 2500 K, the dynamic range required of the detector is less than 10. An additional detector operating at 420 nm and otherwise having the same sensitivity and dynamic range is able to measure temperatures between 1970 and 2250 K while retaining the accuracy of the shorter wavelength detector. Thus, by using several individual different-color pyrometers, the entire temperature range of interest can in principle be measured with high accuracy, assuming only marginal knowledge of the emissivity. Thus, a multicolor pyrometer is considered here to be a group of single color pyrometers of which only one is used at a time. The PSI multicolor <u>imaging</u> pyrometer is designed around this principle. The procedure used to select the actual wavelengths used in the imaging pyrometer is included in Appendix C.

APPENDIX B

Video Characteristics of the Imaging Pyrometer

The RS-170 Video Standard

The photodetection and data acquisition portions of the PSI multicolor imaging pyrometer utilize state-of-the-art commercial video devices. The term "video" refers specifically to EIA standard, RS-170 format monochrome video data transmission. This format requires that a complete video image, called a "frame" consist of two interlaced video "fields", each field containing 18.5 lines of synchronization information and 244 lines of video data thereby providing a total vertical resolution of 488 lines per frame. As shown in Figure B.1, each field fills the height of the video screen and is displayed in 1/60s (16.67 ms). Horizontal lines of the odd fields are placed in between the lines of the even fields by the interlacing process (which is why each field contains a half-integer number of lines), thereby displaying to the human observer a visual image which is temporally and spatially continuous. It is possible, of course, to interpret each field rather than each frame as a complete image and thereby decrease the data transfer period associated with each image, but only at the expense of vertical resolution. This capability is utilized when operating the imaging pyrometer in the variable integration time mode described in Appendix D.

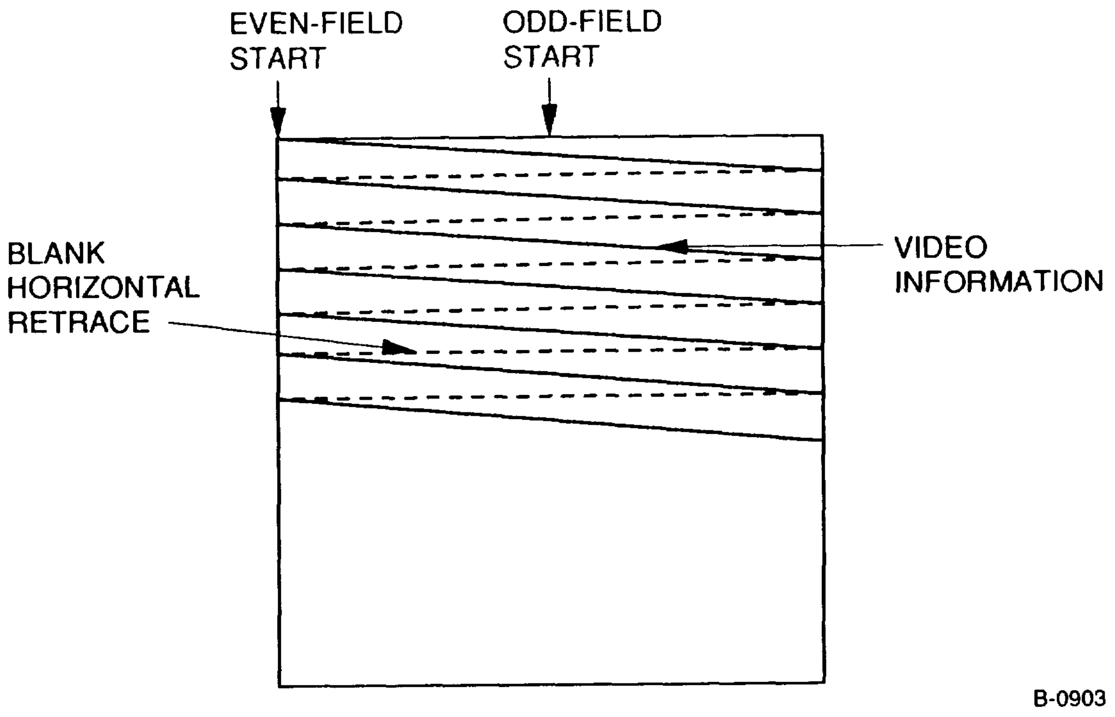


Figure B.1. - Format for displaying the even field of a video frame on a monitor.

Each line of video and sync information is transmitted in the form of a continuously modulated analog signal supplied by the video source (i.e., camera, TV receiver, etc.), known as the composite video signal. An example of the composite signal, showing the sync information and a few horizontal lines of video data, is illustrated in Figure B.2. It is seen that each horizontal line is separated from its predecessor by a horizontal sync pulse, and the fields are separated by vertical sync pulses. Since vertical syncs must be supplied at a rate of 60 Hz, it follows that horizontal sync pulses are supplied at 15,750 Hz, i.e., with a period of 63.5 ms. The horizontal resolution of the video data is limited by the bandwidth of the composite video signal, which in turn may be limited by the image detector, the processing system, or the display monitor. The bandwidth of the Data Translation frame grabber used in the imaging pyrometer is 4.5 MHz, providing resolution of spatial frequencies up to 285/line (which is digitized into 512 pixels/line). This exceeds the resolution provided by the pyrometer's imaging optics.

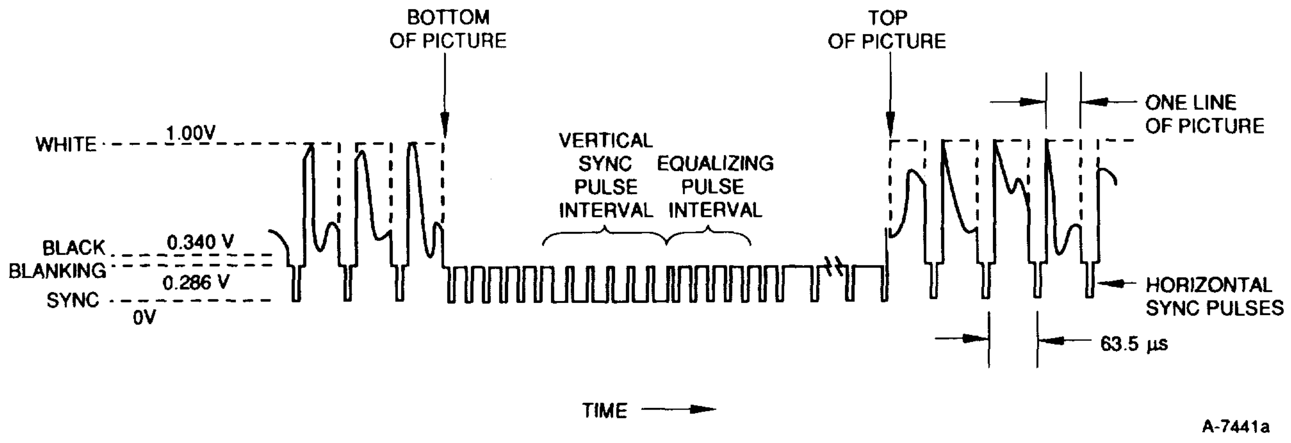


Figure B-2. - Format of a 1-volt p-p RS-170 composite video signal around the vertical retrace interval.

CCD Arrays

The photodetector utilized by the pyrometer is an Amperex NXA1031 Frame Transfer CCD array incorporated into the Sierra Scientific Model MS-4032 camera. A CCD array is an N x M matrix of individual photosites forming vertical columns of depleted photoconductive material separated by "channel stops" of p-type non-photoconductive material. The discrete photosites within each column are defined by potential wells created by semi-transparent surface electrodes. Incident photons generate charge carriers which are trapped in the potential wells.

There are two distinct CCD architectures found in commercial arrays: frame-transfer (FT) arrays, the type used in the imaging pyrometer, and interline-transfer (ILT) arrays. In ILT devices, each vertical column of photosensitive elements is paralleled by a similar charge transfer register which is masked against incoming light. During readout, polysilicon transfer gates shift the integrated charge from each photosensitive column to its corresponding transfer register. The transfer registers are then shifted downward at a fixed clock rate onto a single, horizontal shift register which is then read serially, creating a row by row output signal.

Slightly more than one-half an ILT array's surface area consists of optically masked transfer gates and registers. In contrast, the entire surface of an FT device (except for the narrow channel stops) is photosensitive. In the FT architecture shown in Figure B.3, rather than having a charge transfer register parallel to each photosensitive column, there is immediately adjacent to the photosensitive array a second array of similar construction - but masked from incident light - which serves as a storage array. During readout, the entire signal captured by the photosensitive array is rapidly (~ 160 to 400 μs) moved into the storage section. The photosensitive region then begins to collect the next image while the storage region is read out row by row.

FT arrays can be operated in a fashion that provides interlaced fields from two consecutive integrations on the same photosensitive area, as shown schematically in Figure B.4 for the Phillips NXA1031 array used in the imaging pyrometer. Four electrodes define a photosite in a given column. For the first field integration, the electrode voltages are set up so that the charge accumulation region is centered under electrode 4. The first field is integrated for approximately one-half of the total frame time (~ 16.7 ms) and is then quickly shifted into the storage region. For the second field integration, the electrode voltages are set up so that the charge accumulation region is centered under electrode 2. This, in effect, creates a new "line" located between two lines of the previous field. During the second field integration period, the first field is read out via horizontal registers located at the bottom of the storage section. At the end of the second field integration period, the accumulated charge is quickly swept into the cleared storage area and integration of the first field of the next image is begun. In this manner, an interlaced signal with effective resolution of 488 lines is constructed from an array with only 244 active lines per field.

Frame Transfer CCD arrays are subject to the phenomenon known as vertical smear, which results in a loss of clarity when observing scenes having exceptionally large contrast. Vertical smear is caused by the movement of pixels in each column downwards through other exposed pixel sites during transfer to the storage area. Since new charge is continually generated even during the transfer process, if a spot in one column receives enough light to generate a measureable number of photoelectrons during the transfer time, it will be observed in all pixels along that column, thereby forming a vertical stripe. The brightness of the stripe relative to that of the spot itself can be calculated using equations found in the CCD product literature, which is appended into the Sierra Scientific camera manual.

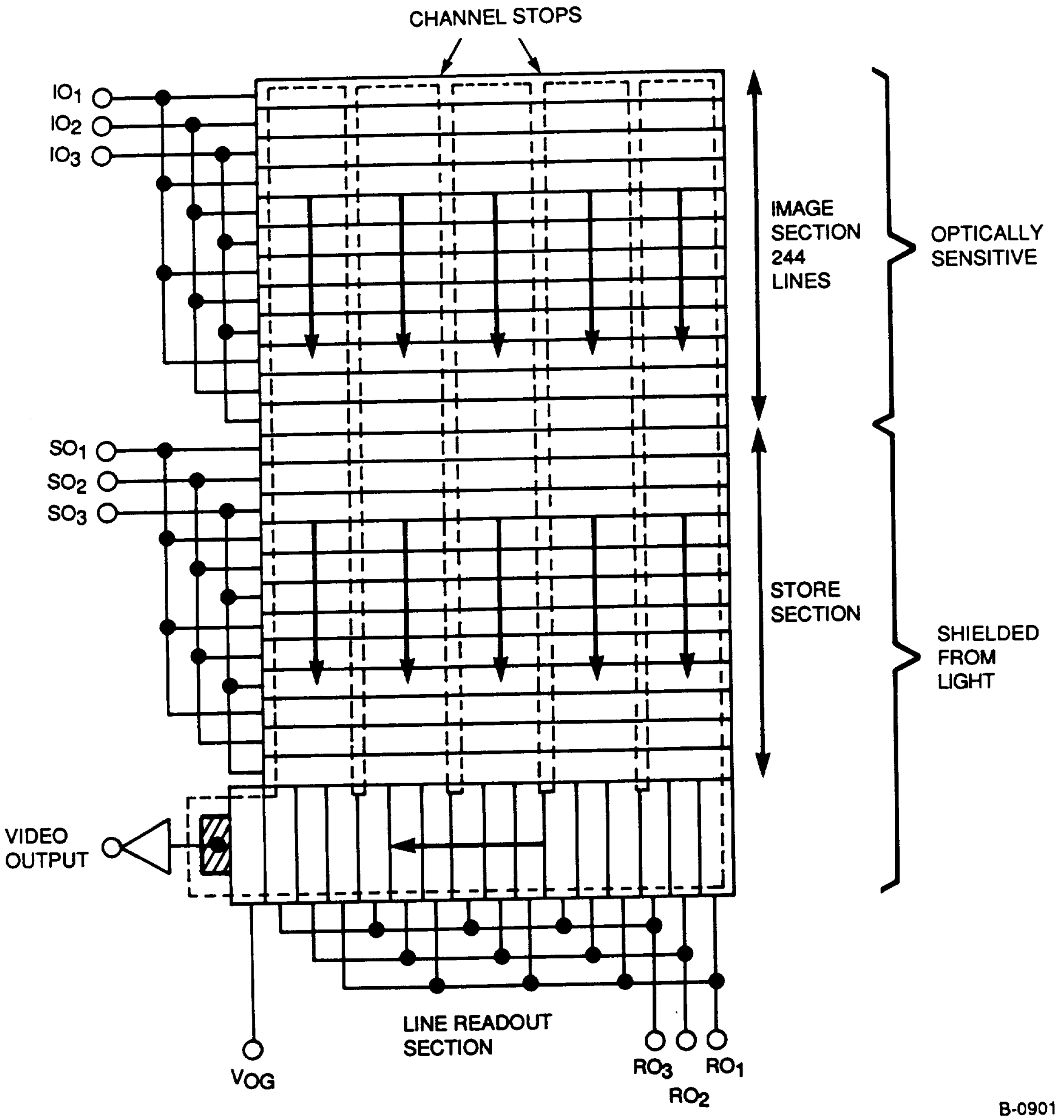


Figure B-3. - Frame transfer CCD array architecture.

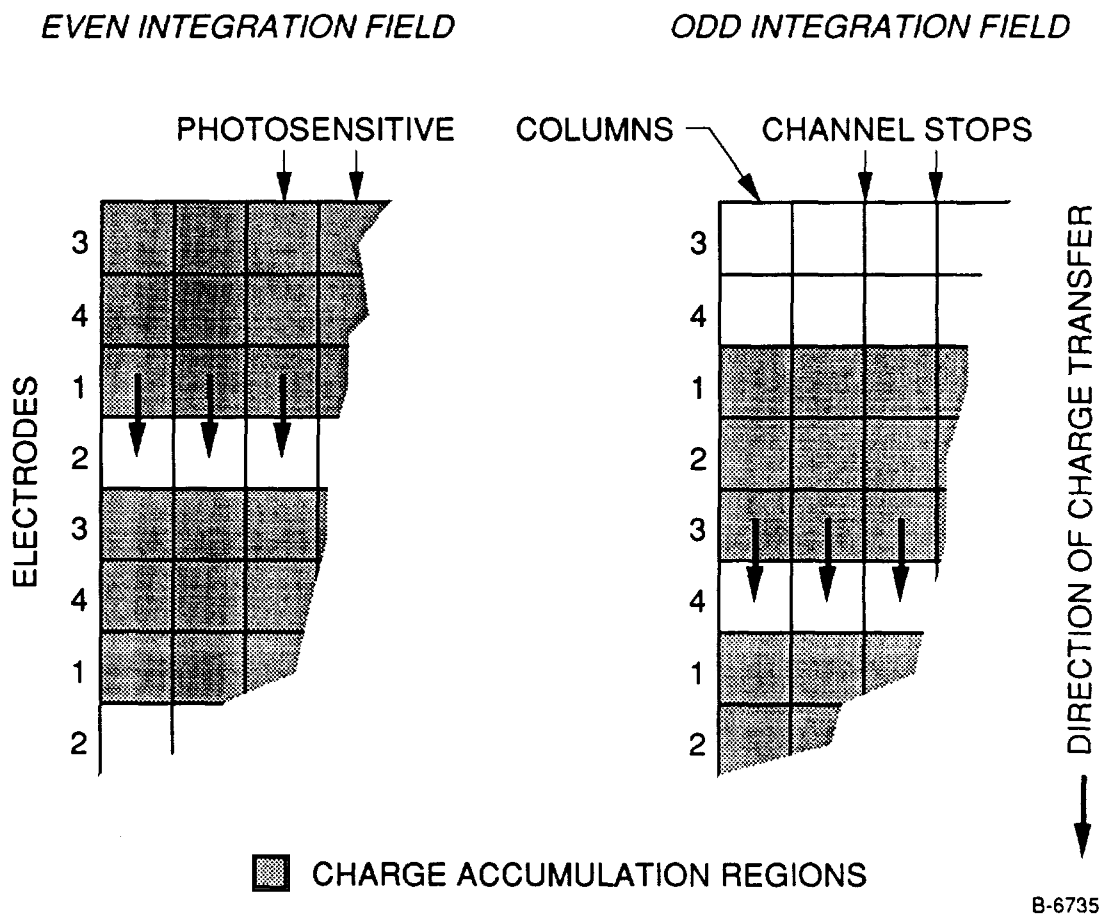


Figure B.4. - Schematic illustration of frame readout sequence for a Phillips Model NXA1031 frame transfer CCD array.

CCD Optical Response Characteristics

One measure of the quality of an imaging device is its sensitivity, which is determined primarily by the number of photoelectrons that must be generated to overcome the noise inherent to the detector. The noise may be thought of as a random number of electrons created by any mechanism other than the reception of photons. More specifically, the noise may be defined as the statistical variation in the output signal measured among many apparently identical realizations of the same sequence of events. The electron well depth in a typical CCD photosite is equivalent to a few hundred thousand electrons, and the intrinsic thermally-induced noise is typically on the order of a few hundred electrons rms. Thus, with careful attention to proper pre-amplification, the FT array is in principle capable of an SNR approaching 1000:1 (60 dB). However, additional noise pickup in most CCD-based cameras limits the SNR to less than 300:1 (50 dB), corresponding to approximately 1000 electrons noise. Using a typical quantum efficiency of 40%, about 2500 photons are required to equal the intrinsic noise of an FT array. (The noise in the imaging pyrometer has been measured to be less than 2040 photons at a wavelength of 630.5 nm, as is discussed below.)

Additional noise may be introduced if the video output signal is supplied to an input device which digitizes it. A device which accepts the video output signal and formats the data into an array which can be processed and displayed by a computer is known as a frame-grabber. In virtually all commercially available frame-grabbers, the digitization is asynchronous with respect to the horizontal clock frequency of the output device. In other words, at the beginning of each video line, the frame-grabber begins to digitize the signal at a pixel clock rate which is independent of the output device's pixel clock. Both clocks operate at a speed which generates or reads one line of video signal in 63.5 μs, independent of the number of individual pixels. For example, the DT-2861 frame grabber used in the pyrometer creates a 512 column frame from the continuous analog signal output by a 610 column NXA1031 CCD detector, disregarding the fact that the analog signal was originally generated by 610 discrete elements. Unfortunately, in the readout of each pixel from the image detector array, a small fixed-phase clock transient is injected onto the video signal. A synchronous A/D converter, i.e., one which digitizes the input at the same rate as the output, would sample this transient at the same relative phase in each pixel, and would thus see it as a constant dc offset that could easily be subtracted. In an asynchronous A/D, however, the digitization window will drift with respect to the clock transient from pixel to pixel, making subtraction of the transient signal more difficult. This spurious signal is thus usually considered to be an additional noise source, though it is not truly random. The magnitude of this noise is critically dependent on the specific camera/frame-grabber combination, but can be expected to be equivalent to about 1000 detector electrons rms. Its effect can occasionally be seen in the video output of the pyrometer as a faint grid of pixels that are one gray level brighter than their surroundings.

To determine the sensitivity of the Sierra Scientific camera/DT-2861 frame-grabber system, its responsivity, defined as the average output voltage signal measured in response to a known input, was first measured by exposing the detector, through an interference filter, to a standard source of spectral irradiance. This source was essentially a quartz-halogen tungsten filament lamp calibrated by Optronics Inc. to NBS traceable standards when energized with a precise current. The camera's CCD detector was located 200 cm from the source. The filter had a central wavelength of 630.5 nm and a bandwidth of 11.0 nm. According to the calibration report, at a point located 50 cm from its center the bulb supplied a spectral irradiance of 0.528 $\mu W/cm^2$-nm at 630.5 nm. From the inverse-square law, the irradiance at 200 cm was 0.033 $\mu W/cm^2$-nm. Since the energy per photon was $hc/\lambda = 3.15 \times 10^{-19}J$, and the area of one pixel is $1.84 \times 10^{-6}\ cm^2$, this irradiance corresponds to a photon flux of 1.21×10^6 photons/pixel-s.

Figure B.5 plots, as both a function of integration time and average number of incident photons, the change in the camera output, relative to the dark signal, when exposed to the light source. To obtain these data, a "dark" frame, i.e., one obtained when the camera was shielded from light, was subtracted from each "bright" frame and a histogram of the number of pixels at each digitization level calculated. The data points shown are the averages of the peak values of six such histograms for each integration time. The data of

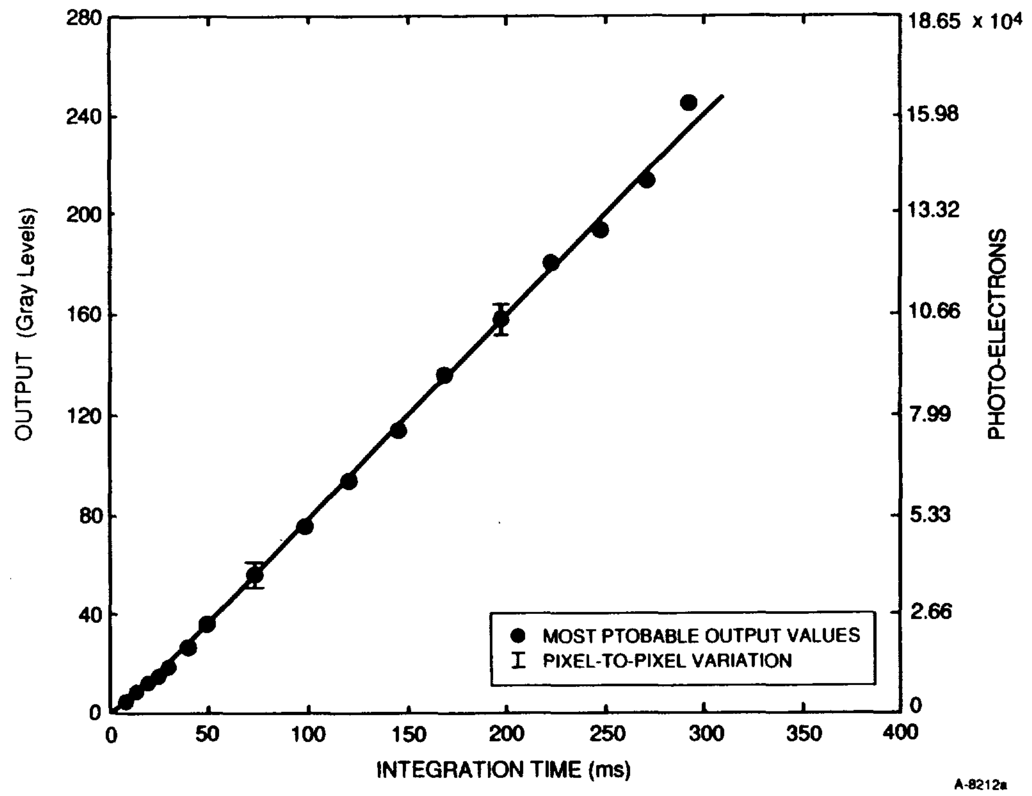


Figure B.5. - Responsivity curve.

Figure B.5 form a very nearly straight line with a slope of 6.74 x 10^{-4} digitizer steps per photon, or 1480 photons (@ 630.5 nm) per step. Close examination of Figure B.5 shows that there is some slight curvature of the output at small integration times, so that the data pass through the origin. Figure B.6 shows similar calibration curves for each four pixels selected because they each exhibit a dark current that is dramatically different from the others, as plotted in Figure B.7. Despite these dark current variations, all four pixels have responsivities that are identical to within approximately ± 3 percent, and agree with the average camera responsivity to the same level of accuracy.

The noise of each of the four pixels was evaluated by calculating the standard deviations around their average signal values determined from over 1000 successive measurements. The data indicate that the total noise at any pixel is roughly proportional to that pixel's average measured signal value, regardless of whether signal is due to dark current or to exposure to light. For integration times less than 200 ms, the total noise roughly fits the equation

$$\Delta V \approx 0.45 + v/275 \qquad \text{(B.1)}$$

where V is the measured gray level. Since the maximum value of V is 255, at camera saturation the noise is about 1.38 digitizer steps, corresponding to 2040 photons. Of more significance is the noise near the threshold of sensitivity, where V = 0 and ΔV = 0.45, corresponding to 670 photons.

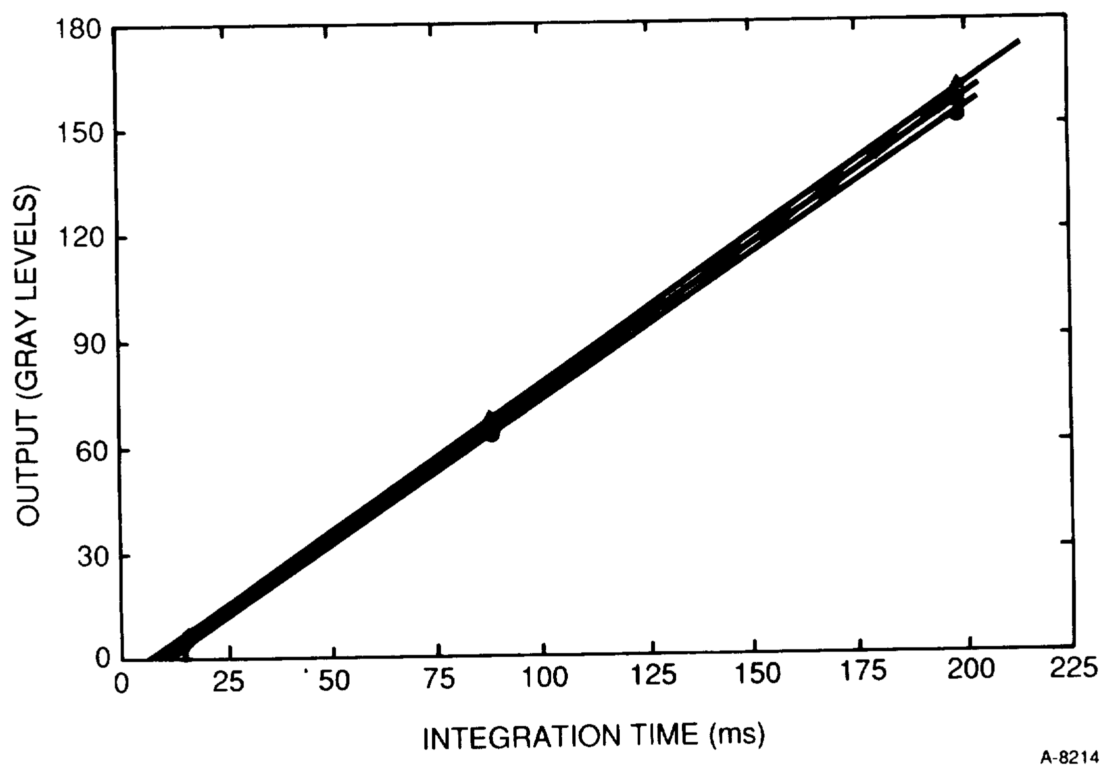


Figure B.6. - Relative responsivities of four selected pixels.

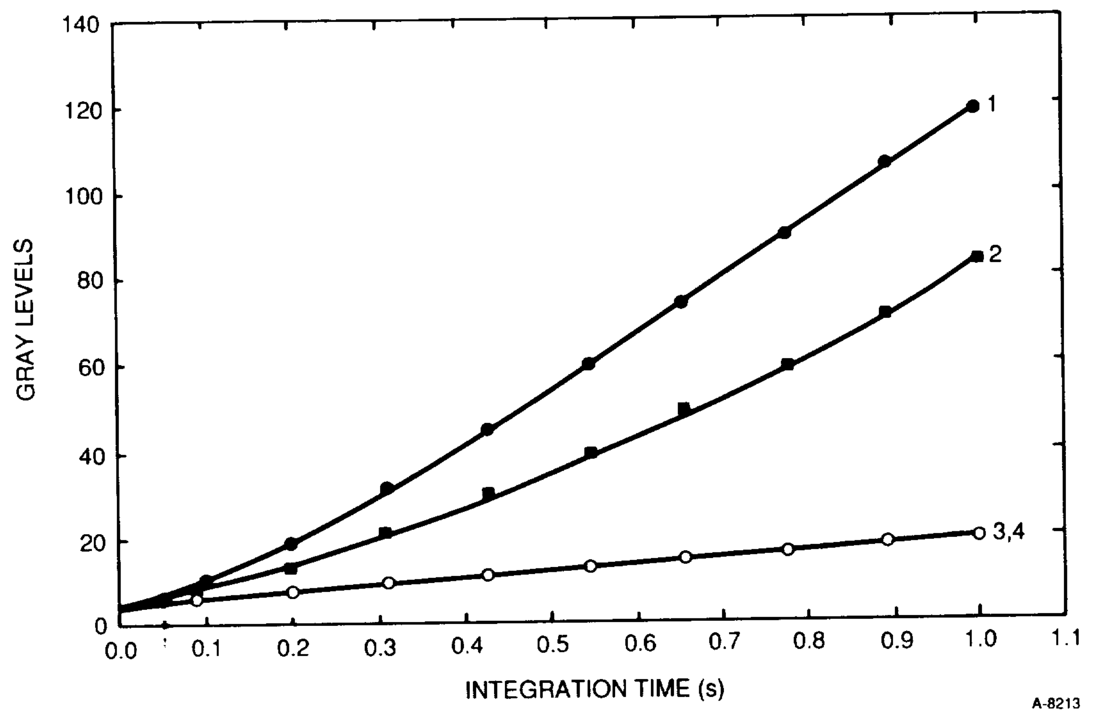


Figure B.7. - Dark currents of the four pixels used in Figure B.6.

APPENDIX C

Imaging Pyrometer Optical System

C.1 Design and Layout

The imaging pyrometer was designed to accurately measure surface temperature distributions of objects having diameters of approximately 2 mm moving within a spherical volume of about 1 cm diameter. Discussions with NASA personnel early in the design process indicated that the spatial resolution offered by the NXA 1031 CCD detector is more than adequate for determining temperature gradients, even when heated samples are imaged onto the array at 1/5 actual size, assuming of course that the resolution of the imaging optics would be roughly equivalent to the detector resolution. (See, however, the discussion of MTF limitations below.) The 4.5 mm x 6.0 mm detector area is large enough to accomodate up to six such non-overlapping images. This capability is utilized in the pyrometer to observe the heated sample at six wavelengths simultaneously, using no moving parts.

The optical system built to provide the six images is illustrated schematically in Figure C.1. The heated sample is located in a plane 360 ± 5 mm from the outer principle plane of the pyrometer's 360 mm focal length objective lens. Radiation from the object is nearly collimated by the objective and then projected into a color separator comprised of a series of

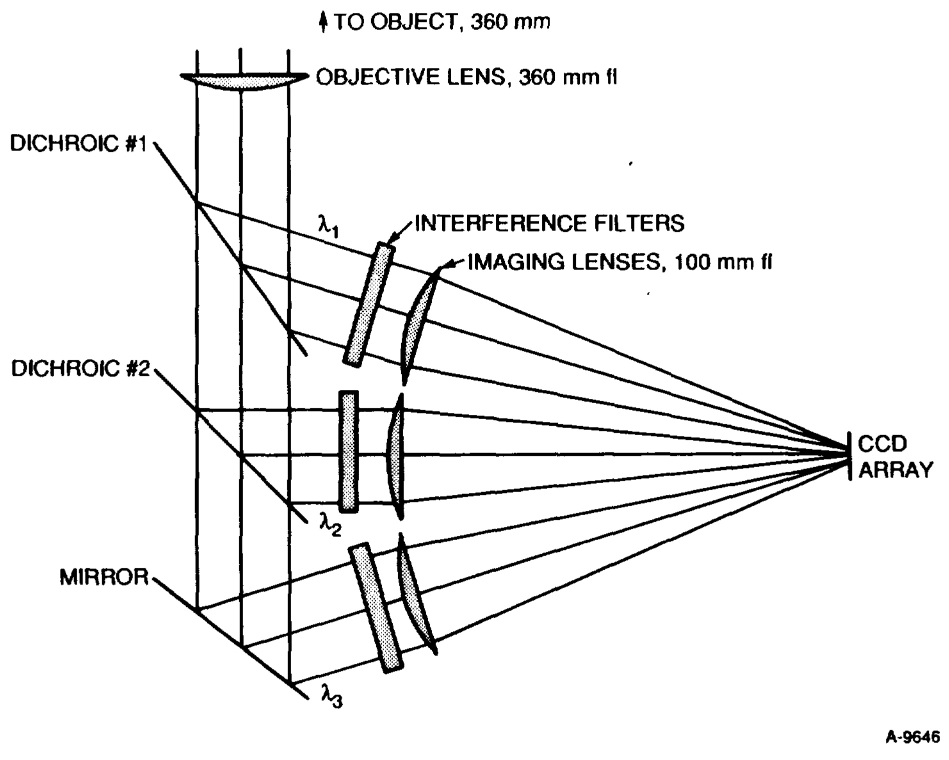


Figure C.1. - Top view of color separator.

dichroic beamsplitters, three of which are shown in Figure C.1. The first beamsplitter ideally reflects all radiation at wavelengths greater than λ_1 while transmitting all shorter wavelengths. The second beamsplitter similarly reflects all radiation at wavelengths longer than λ_2 (of which there is ideally none longer than λ_1) and transmits the rest, which is reflected by the third mirror. The three reflected beams are transmitted through interference filters that select the desired narrow bands of wavelengths from the relatively broad bands reflected by the beamsplitters, and thence through 100 mm focal length lenses that image the sample onto the CCD array located at the focal plane of the lenses. To provide all six images, a fourth dichroic beamsplitter is inserted ahead of the three shown. It reflects a band of wavelengths downwards to a second row of beamsplitters. The reflections from this second layer are projected similarly onto the CCD, as illustrated by the side view of Figure C.2. The optical collection efficiency of the system is limited by the 25 mm diameter imaging lenses to about F/14, but to reduce optical aberrations and vignetting, a 20 mm aperture has been placed at the objective lens limiting the speed to F/18. The overall system magnification is 0.28, determined by the ratio of the focal lengths of the objective lens to the imaging lenses.

The pyrometer's optical head is constructed on two symmetrical baseplates, each holding one of the two color separators and one of which is illustrated in Figure C.3. The drawing is oriented so that the objective lens is mounted in a bracket at the top of the L-shaped configuration, and the CCD camera is supported by a bracket located at the right edge. Progressing from the objective lens, the platform holds a triangular mount supporting the dichroic beamsplitter that separates the short wavelengths from the long and directs them downwards to the second color separator. Beyond this beamsplitter are the two dichroic beamsplitters and an aluminized mirror, this group comprising the long-wavelength color separator. Each of these 5 cm (square or circular) components is mounted on a 2.54 x 2.54 cm adjustable mirror mount by means of small holders. Each of the three color-separated beams then are directed

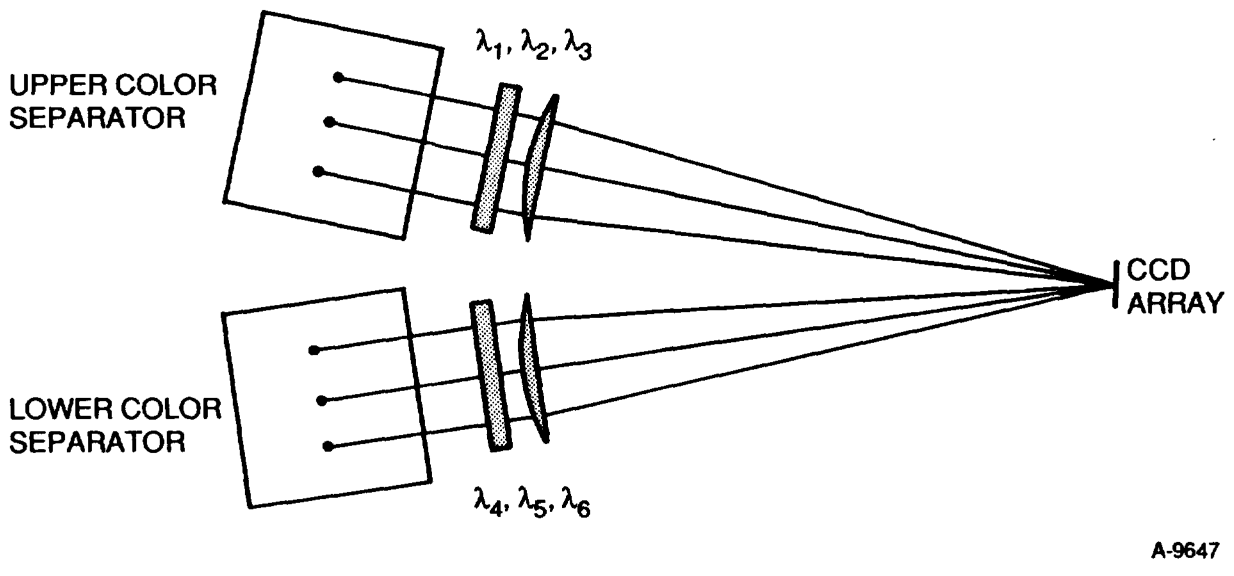
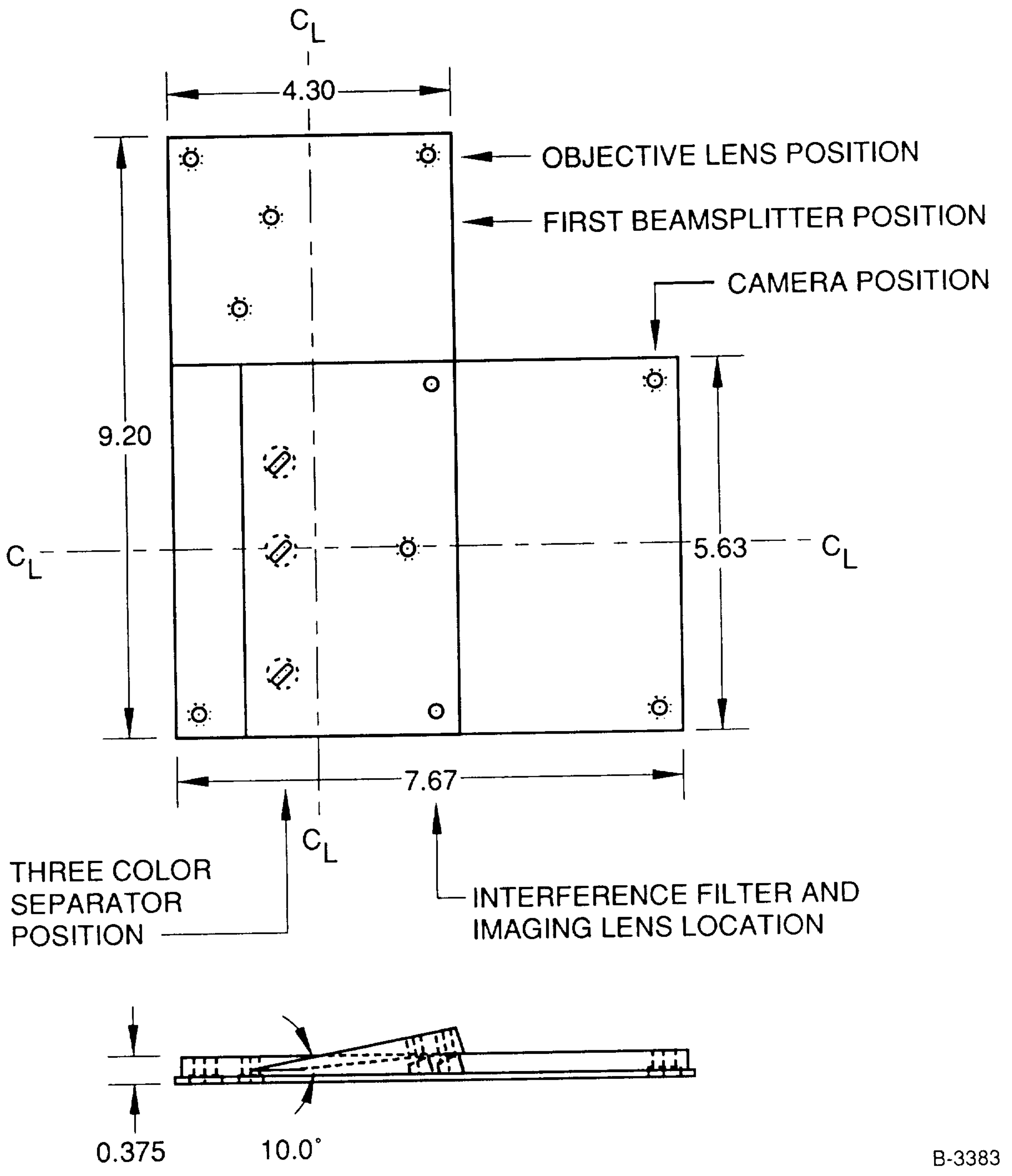

UPPER COLOR SEPARATOR
$\lambda_1, \lambda_2, \lambda_3$
CCD ARRAY
LOWER COLOR SEPARATOR
$\lambda_4, \lambda_5, \lambda_6$
A-9647

Figure C.2. - Side view of color separator.



C_L
4.30
OBJECTIVE LENS POSITION
FIRST BEAMSPLITTER POSITION
CAMERA POSITION
9.20
C_L
5.63
C_L
7.67
C_L
THREE COLOR SEPARATOR POSITION
INTERFERENCE FILTER AND IMAGING LENS LOCATION
0.375
10.0°
B-3383

Figure C.3. - Baseplate configuration.

through a narrow bandpass interference filter and imaging lens, both glued into opposite ends of a screw mount that fits into a bracket. Proper centering of the beams in the lens holder is achieved by adjusting the mirror mounts, and fine focusing is facilitated by turning the lenses in their screw mounts.

The six optical paths are aligned so that the centers of the images are positioned on the CCD as illustrated in Figure C.4. Each center is 1.5 mm from its neighbors, meaning that objects 1.5/0.28 = 5.3 mm in diameter can be observed without the images overlapping each other. Since the image centers are also at least 1.5 mm from the edge of the detector, each image is a projection of the full 1 cm field of view.

Because the images are projected onto the CCD detector from off-normal angles of incidence, they suffer some minor distortions. As illustrated in Figures C.1 and C.2, each of the two rows images is projected at a ±10 deg angle relative the detector vertical, and the outer images are projected at ±17 deg relative to the horizontal. The net angle of incidence for the outer images is therefore about 20 deg, distorting the image of a circular object into an ellipse with major-to-minor axis ratios of 1.06. This minor distortion does not affect the temperature measurement accuracy, since the effect of increased image area has been included in the calibration. Furthermore, because the planes containing the optical axes of the upper and lower rows of images are tilted 20 deg relative to each other, features within an object seen in the upper row of images are rotated by 20 deg relative to the lower row. Similarly, as the center of the object moves along a horizontal line centered vertically on the optical axis, the upper and lower rows of images move along 10 deg lines relative to the horizontal. Large horizontal motions can make the images appear to squeeze together or move apart.

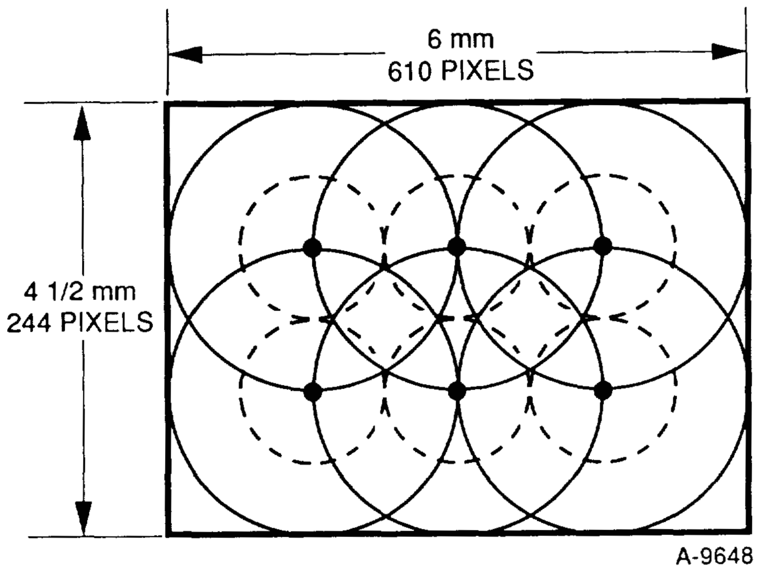


Figure C.4. - Arrangement of images on detector. Dashed circles indicate maximum size of object that can be observed wihtout image overlap. Solid circles indicate field-of-view available to each image.

All of these effects can be expressed mathematically as relationships between the coordinates of a pixel in one of the six images, and the coordinates of the corresponding pixel in the other five images. Denoting the longest wavelength image, which appears in the lower left hand corner of the monitor, as image #0, and counting the remaining images in counter-clockwise order, then when the pyrometer is properly aligned the following relationships hold:

$$\begin{aligned}
x_1 &= 0.956\ x_0 + 133.59 & y_1 &= y_0 \\
x_2 &= x_0 + 256 & y_2 &= y_0 \\
x_3 &= 0.940\ x_0 + 0.357\ y_0 + 149.28 & y_3 &= -0.327\ x_0 + 0.940\ y_0 - 98.84 \\
x_4 &= 0.899\ x_0 + 0.342\ y_0 + 31.53 & y_4 &= y_3 \\
x_5 &= 0.940\ x_0 + 0.357\ y_0 - 106.72 & y_5 &= y_3
\end{aligned} \tag{C.1}$$

where (X_i, y_i) are the (horizontal, vertical) coordinates of a pixel.

Proper alignment means that the optics are adjusted so that the centers of the six images lie at the following coordinates on the video monitor:

(128,160)	(256,160)	(384,160)
(128,320)	(256, 320)	(384,320)

The alignment procedure is described in Section 4 of the operator's manual.

C.2 Wavelength Selection Procedure and System Performance

a) Wavelength Selection - Accuracy Versus Precision

The six wavelengths used in the imaging pyrometer have been selected to provide high measurement accuracy and precision while spanning a temperature range from less than 1000 K to greater than 2500 K. The longest wavelength was selected to provide measurement of the lowest temperature possible with this system, determined as follows: The minimum temperature detectable at any given wavelength is that which, during the exposure time, generates a signal equal to the detector's rms noise. In general, as the wavelength increases the associated minimum detectable temperature decreases, and conversely as the temperature decreases the optimum wavelength for measuring it increases. However, at long wavelengths the quantum efficiency of the detector decreases with increasing wavelength, resulting in the existence of a temperature below which no signal can be detected regardless of the wavelength. This is the minimum measureable temperature and, for our system, has been calculated to be 888 K at a wavelength of 930 nm assuming an exposure time of 50 ms. The maximum temperature observable at this wavelength, i.e. that which saturates the detector, is 1302 K.

All of the other colors were chosen to provide a precision better than 0.2 percent during a 16.67 ms exposure at a minimum temperature equal to the maximum temperature observable with the next longest wavelength. The precision of the temperature measurement is, in essence, determined by the 8-bit precision of the video frame grabber, in which the minimum resolvable change in signal is $1/(2^8-1) = 1/255$ of the signal at saturation. Each 1-bit signal change is called a gray level. (The pyrometer has been configured so that, as described in Appendix B, the rms system noise is approximately one gray level. Thus, the precision as defined here is the same as the Noise Equivalent Temperature Change.) Because the signal generated at a specific wavelength increases much more rapidly than linear with increasing temperature, the precision of the measurement also increases with temperature. That is, at higher temperatures a smaller change in temperature is required to produce an incremental step in the gray level than at lower temperatures. Thus, the minimum temperature at which a specified precision can be achieved may correspond to a gray level greater than unity, and therefore is higher than the minimum detectable temperature. Each color therefore has both a wavelength and a minimum gray level specified for optimum precision. The selected colors, the range of temperatures theoretically spanned by each color while providing a measurement precision of 0.2 percent, and the ranges measurable regardless of precision are shown in Table C.1. The ratio of temperature uncertainty to the emissivity uncertainty at the peak temperature of each color is also tabulated. The temperature ranges shown in Table C.1 assume a sample emissivity of 0.5. Figure C.5 graphically displays the range of temperatures theoretically measurable at each of these wavelengths as a function of emissivity. In the shaded regions the upper range of one filter overlaps with the lower range of the filter of next shorter wavelength. Note that the lower temperature limit for each color can be reduced by increasing the integration time. Furthermore, the precision can be made even better by incorporating user knowledge of the approximate sample temperature and operating the pyrometer in a mode that selects a wavelength longer than normal, making use of the system's variable integration time feature to accomodate the increased photon flux. The measurement would, of course, suffer a loss of accuracy in the face of uncertain emissivity. Automated software to perform this function has not yet been developed.

TABLE C.1. PREDICTED PYROMETER CHARACTERISTICS

Wavelength (nm)	High Temperature (K)	Low Temperature @ 0.2% Precision	Absolute Minimum Temperature	$\Delta T/T$ / $\Delta\varepsilon/\varepsilon$
930	1302	1119	888	0.08
750	1506	1298	1050	0.08
590	1728	1505	1244	0.07
480	1971	1726	1445	0.07
420	2249	1970	1650	0.07
370	2562	2243	1878	0.07

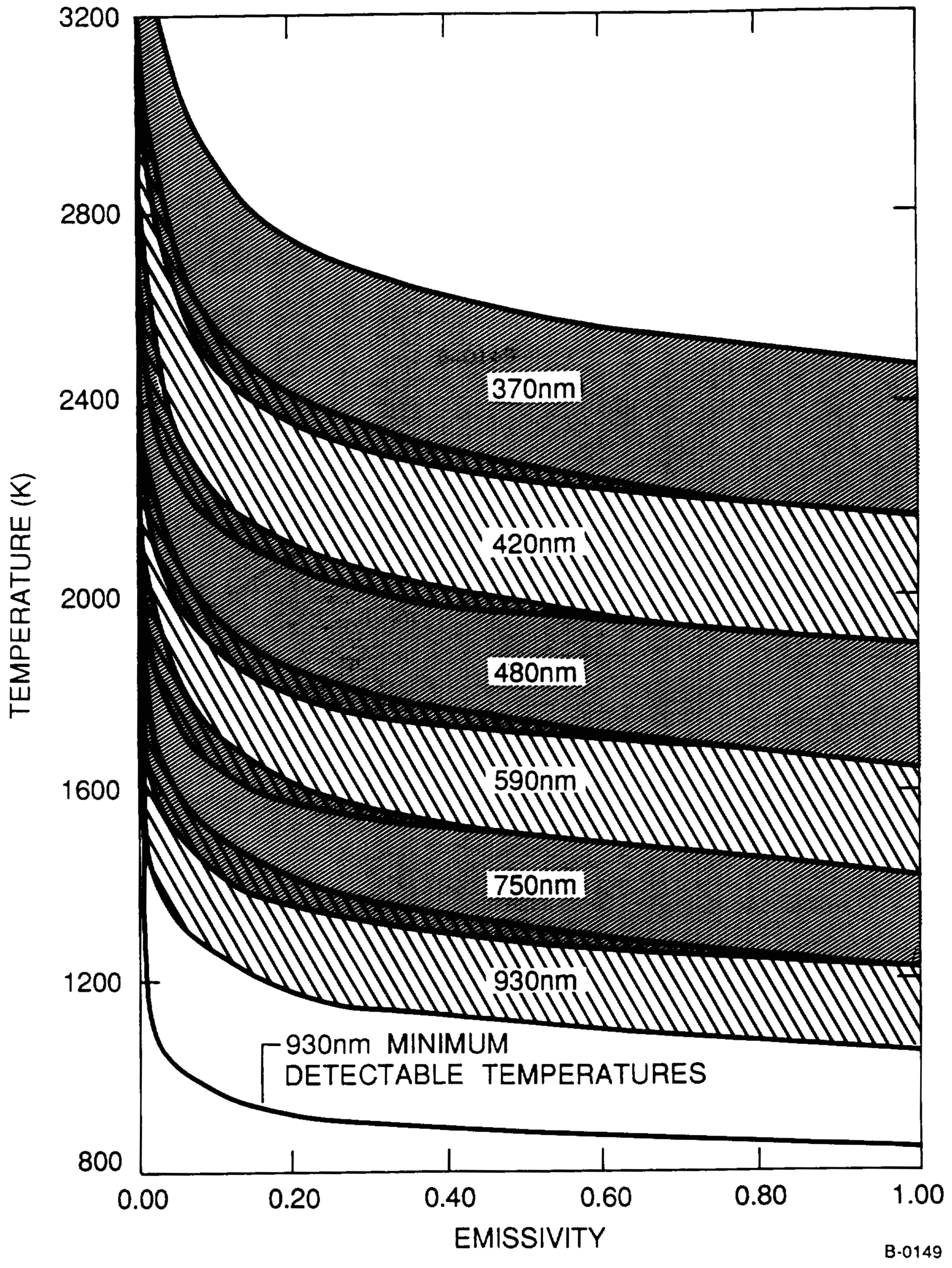


Figure C.5. - Predicted temperature ranges spanned by each color as a function of sample emissivity.

The actual wavelengths used in construction of the pyrometer, reported in Appendix E, vary slightly from those of Table C.1. The differences are due to the need to match the spectral regions over which the reflections from the dichroics are nearly independent of both wavelength and angle of incidence with the bandpasses of the interference filters. Furthermore, the shortest wavelength dichoric was found to absorb substantially, resulting in sensitivity to higher temperatures than anticipated. This effectively results in a loss of precision at the low end of its temperature range.

b) Optical Performance

One reason for using an imaging pyrometer (as opposed to a single point detector) is to have the ability to accurately measure temperature gradients. The accuracy with which gradients can be determined is determined primarily by the precision with which the radiance differences between neighboring pixels can be measured. (Note that the absolute accuracy of the measurement is relatively unimportant for measuring gradients, since errors in the temperature measurement resulting from emissivity or calibration uncertatinties are likely to be nearly uniform across the surface of the sample.) As discussed above, the precision is defined as the change in temperature required to change the output of the frame grabber by one gray level, divided by the actual temperature. The imaging pyrometer has been designed so that the worst precision that will be experienced throughout its measurement range is less than 0.2 percent. Thus, temperature differences between neighhboring pixels as small as than 0.2 percent of the actual temperature can, in principle, be discerned. In practice, however, the transmission characteristics of the optical system affect the accuracy with which temperature gradient information from the object is transmitted to and received by the camera.

The problem is shown pictorially in Figure C.6. Assume that there is a step change in the radiance emitted by the object as a function of position, as indicated in the left hand side of the figure. As shown in the right hand

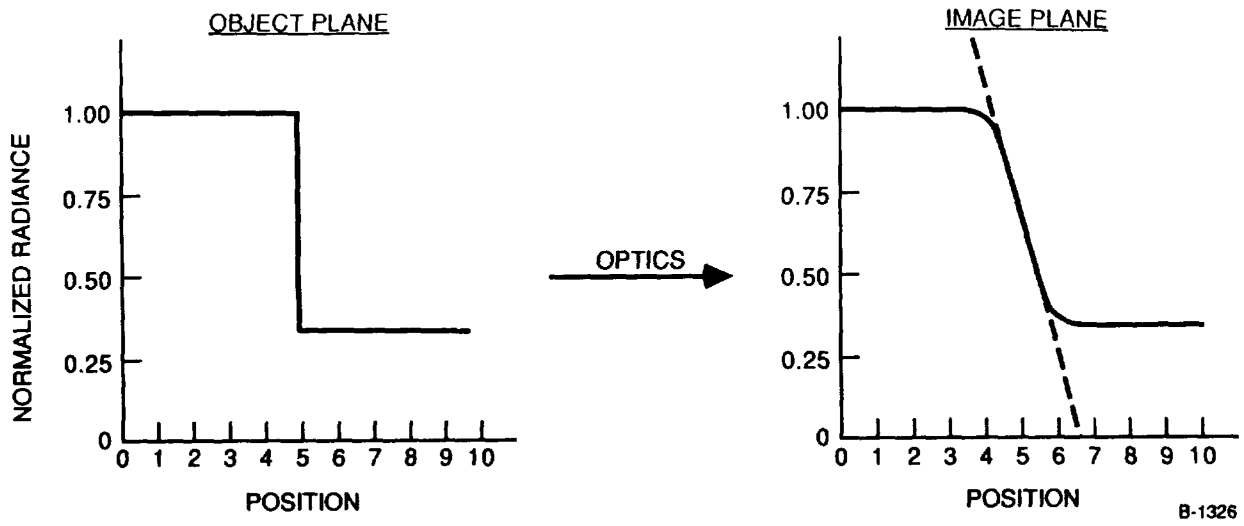


Figure C.6. - Optical system spreading of a step change in object radiance.

illustration, after transmission through the optical system the step change is transformed by diffraction and aberrations into a radiance profile that appears to vary smoothly with position. This smearing is usually quantified by the point spread or modulation transfer functions (which are Fourier transforms of each other). However, as discussed in the following, we quantify this spreading in a fashion which can be directly related to the gradient measurement problem.

During measurement of temperature, it is unlikely (indeed, impossible) that a step change in radiance will be encountered - actual changes will occur over finite, albeit small, distances. Nevertheless, as suggested by Figure C.7, changes in radiance which in reality are spread out over small distances are observed to spread into larger distances after transmission through the optics. However, as the gradient gets smaller, i.e, the radiance change is spread out over a larger distance, the difference between the actual and observed gradient also gets smaller. Indeed, there is some distance over which the radiance change can be spread so that the measured gradient is indistinguishable from its actual value. If radiance changes are observed to be spread over distances larger than this value than it can be assumed that the measurements are faithful reproductions of the actual situation. Conversely, if changes appear to occur over shorter distances, gradients that are actually steeper than measured could be present.

By using a knife edge illuminated uniformly from behind to measure the point spread function of our optical system, the distance over which temperature gradients can be accurately reproduced has been calculated. Figure C.8 illustrates the measurement procedure. The knife edge provided a step function change in brightness of the otherwise uniform radiation emitted by the blackbody. It was positioned so that the best possible image of it was

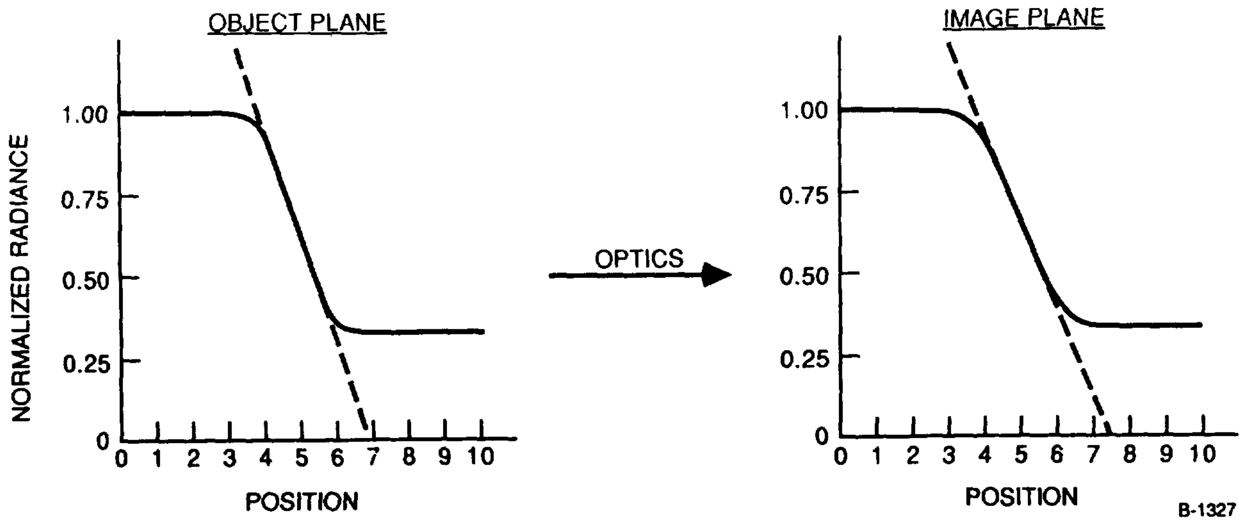


Figure C.7. - Optical system spreading of a gradual radiance change.

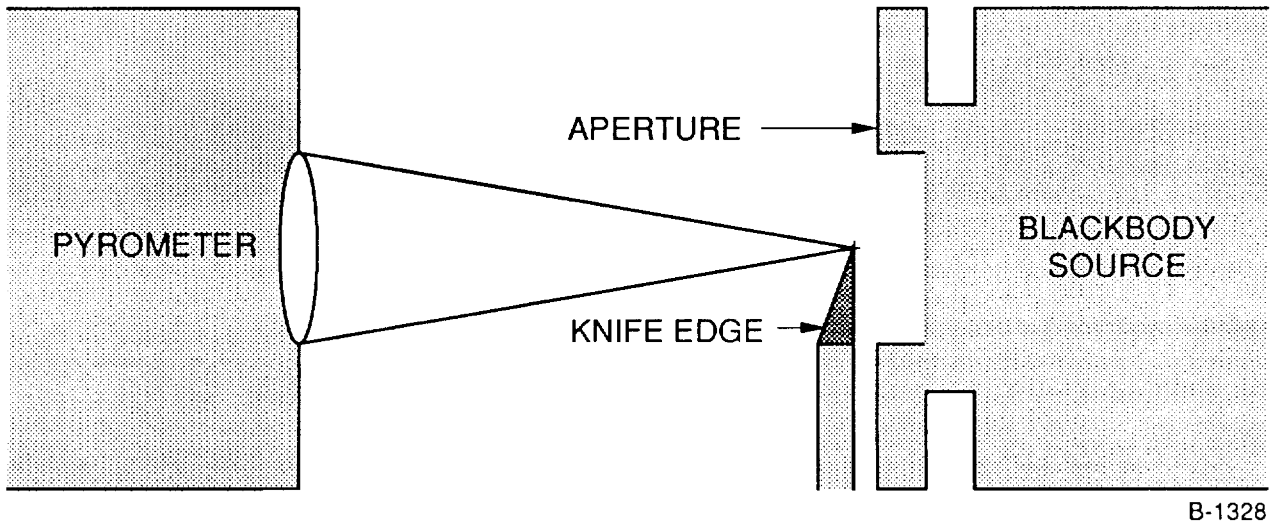


Figure C.8. - Schematic diagram of apparatus used for measurement of point spread function.

projected (six times) by the pyrometer optics onto the camera. The intensity profile along a line perpendicularly crossing one of the six images was measured, and the point spread function calculated from

$$P(x) = \frac{1}{I(x)} \frac{dI}{dx} \quad . \tag{C.2}$$

The measured point spread function was then used to determine the intensity profile $I_m(x)$ that would be measured for an arbitrary actual intensity profile, using a simple convolution:

$$I_m(x) = I(x) \otimes P(x) \quad . \tag{C.3}$$

This calculation was performed by invoking the convolution theorem, thereby converting the convolution procedure to a simple multiplication of the Fourier transforms the two right-hand terms in Eq. (C.3), followed by inverse Fourier transformation. As mentioned above, the Fourier transform of P(x) is the modulation transfer function (MTF) of the optical system. The MTF for the 730 nm image is shown in Figure C.9.

This MTF was used with Eq. (C.3) to evaluate the fidelity with which object intensity profiles having constant gradients could be reproduced. Figure C.10 graphically compares the theoretical actual intensity gradient, normalized by the total change in intensity across the gradient, that would produce a given measured intensity gradient, similarly normalized. Where the actual and measured values are equal, i.e., along the line labelled y=x, the measured intensity is an accurate representation of reality. Figure C.10 shows that this is achieved for normalized gradients smaller than 0.004 μm^{-1}, meaning that gradients that are spread out over distances exceeding about 250 μm are measured accurately.

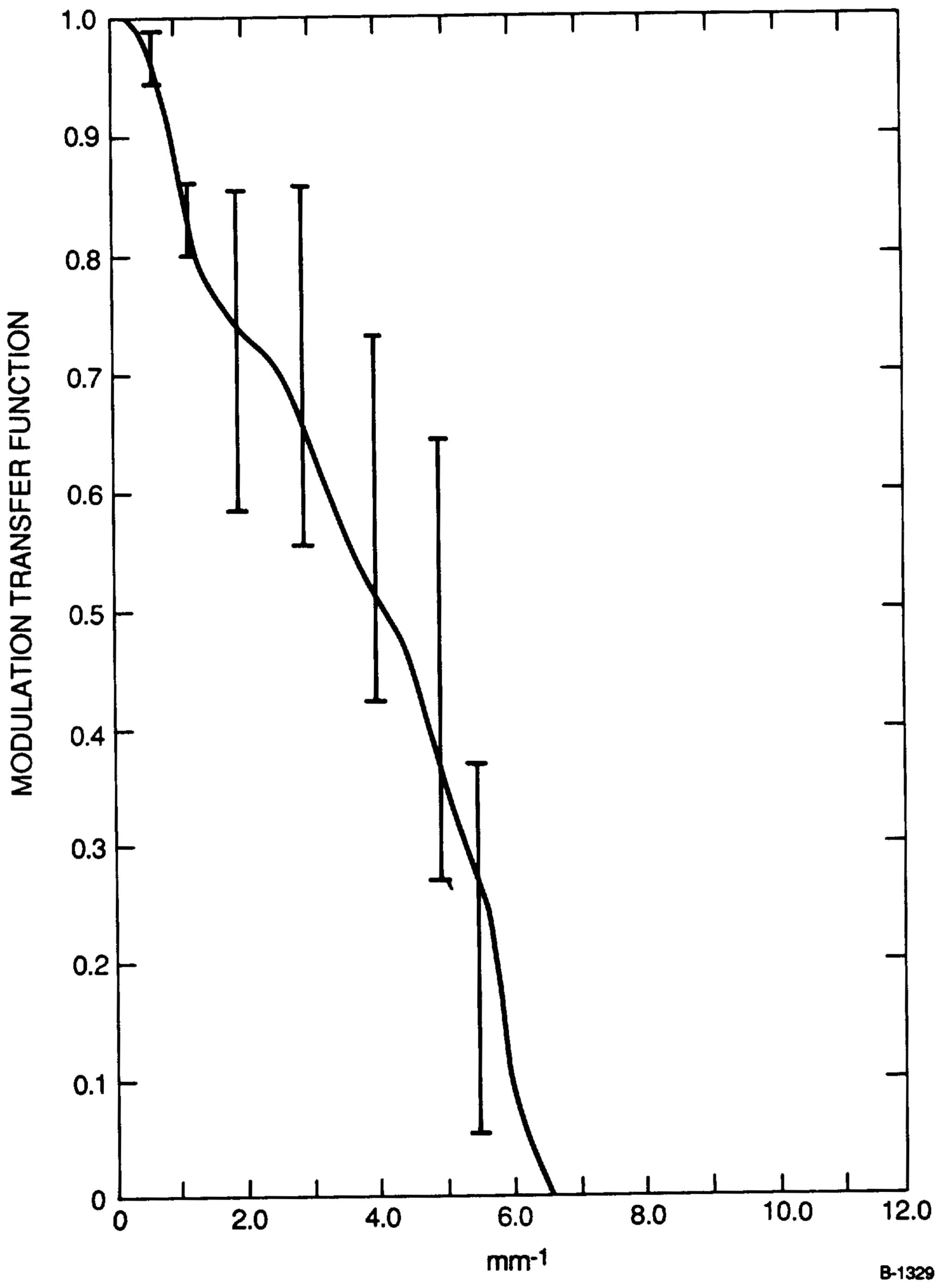


Figure C.9. - MTF of 730 nm image.

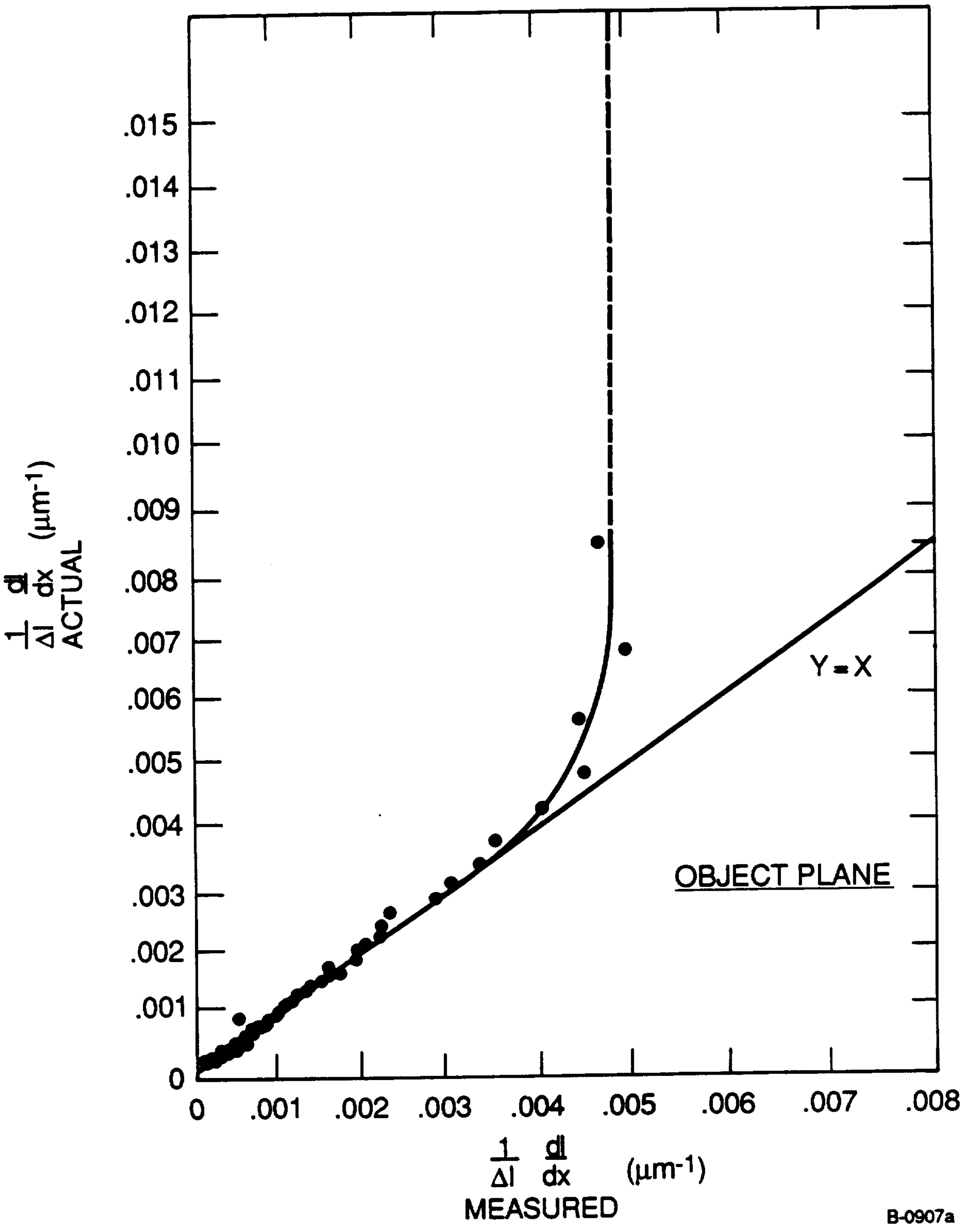


Figure C.10. - Comparison of actual normalized intensity gradients with those that would be measured using the pyrometer.

APPENDIX D

Exposure Time Controller

The multi-color imaging pyrometer uses a Sierra Scientific Model 4032 Frame-Transfer type CCD camera as the optical detector. This camera was chosen over other candidates primarily because of its capability to provide selectable exposure integration times ranging between 1 and 1000 ms. This feature essentially provides for a programmable electronic shutter requiring no moving parts, ideal for use in a space environment. To optimize the use of this feature, the pyrometer is supplied with an interface that allows the exposure time to be controlled by a simple analog voltage signal, which may be provided by the D/A output from the computer, and makes the camera's output in the variable integration mode of operation compatible with the video frame grabber. The operation of this interface is described here.

Interface Requirements

When operating the Sierra Scientific camera in its variable exposure mode, the exposure time is controlled by the interval between two TTL pulses, each having a duration of 150 to 250 μs with the leading edge falling and coincident with the leading edge of a horizontal sync pulse. The first pulse clears the camera sensor. After receipt of this clear pulse, each CCD pixel begins to accumulate charge proportional to the irradiance on its surface, but the camera electronics continues to supply an output of blank RS-170 fields, i.e., fields containing all of the sync information but with the video information suppressed. These blank fields are supplied until receipt of the second pulse. This read pulse causes the camera to interrupt the blank field which it is transmitting, and immediately reset to the top of a new field containing the video information collected during the exposure time. After finishing this field, the camera returns to a waiting state and continues to supply blank fields.

The blank field which is interrupted by the read pulse generally has a duration of less than 1/60s, and therefore does not strictly correspond to the RS-170 format. Because of this, the video receiver (monitor or frame grabber) to which the camera output is attached is momentarily confused until its synchronization circuitry has time to adjust to the change, resulting in a garbled picture. To provide a clear and continuous video picture, the exposure time controller has been designed to supply read pulses that have leading edges which are coincident with the leading edges of the vertical syncs supplied by the camera. Furthermore, so as to acquire and process data as rapidly as possible (which is highly desirable when using the pyrometer in a measurement and control application), the control circuitry has been designed to begin a new exposure cycle during the first blank field after transmittal of the video data. Of course, the timing of the next clear pulse is set up so that the subsequent read pulse occurs only after the desired exposure time and coincident with the vertical sync.

To summarize, the interface which controls the camera exposure timing satisfies the following specifications:

1. Supplies horizontal sync pulses of about 6 μs duration at a rate around 15,750 Hz;

2. Supplies clear pulses of 150 to 250 μs duration with leading edges coincident with the leading edge of a horizontal sync;

3. Supplies read pulses of 150 to 250 μs duration with leading edges coincident with the leading edges of both horizontal and vertical syncs;

4. Begins a new exposure cycle during the field immediately following the video readout; and,

5. Provides for computer control of the exposure duration via an analog voltage level.

Interface Design

The interface circuit that performs these functions is shown schematically in Figure D.1. A timing diagram is shown in Figure D.2 to aid explanation of its operation. The interface operates as follows: At some initial time the user selects an exposure time and resets the system by pressing and holding the reset button for 1 or 2 seconds. At this point, the camera provides only blank

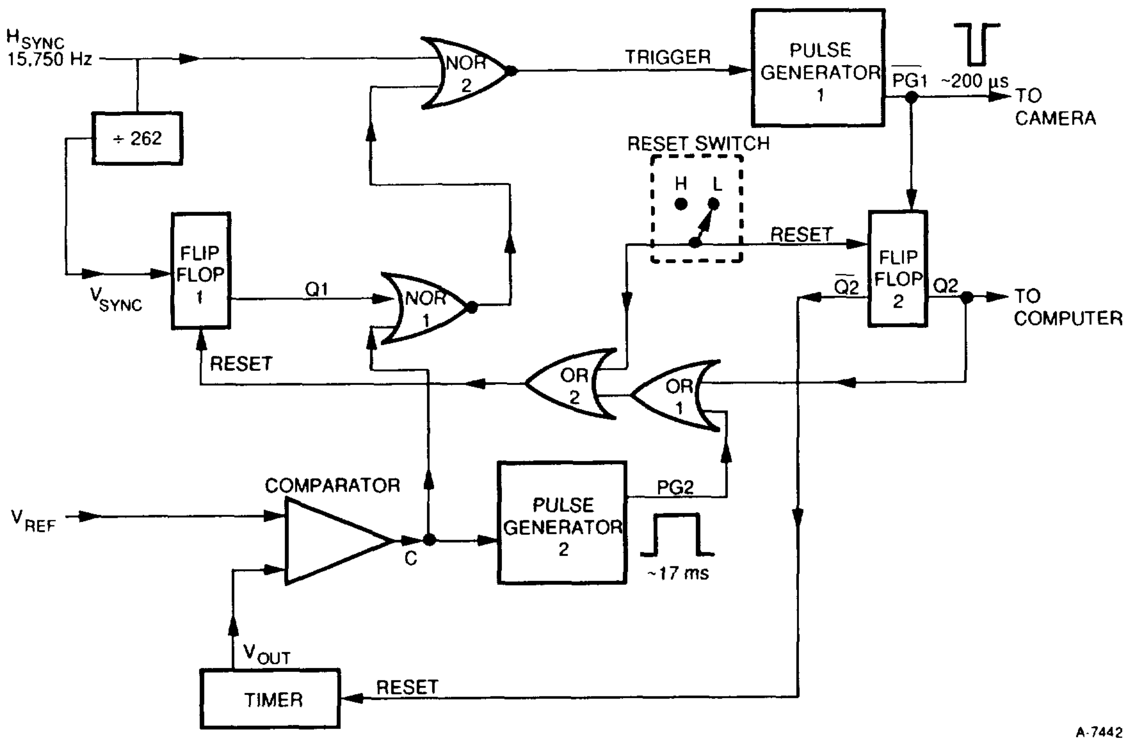


Figure D.1. - Block diagram of camera/computer interface.

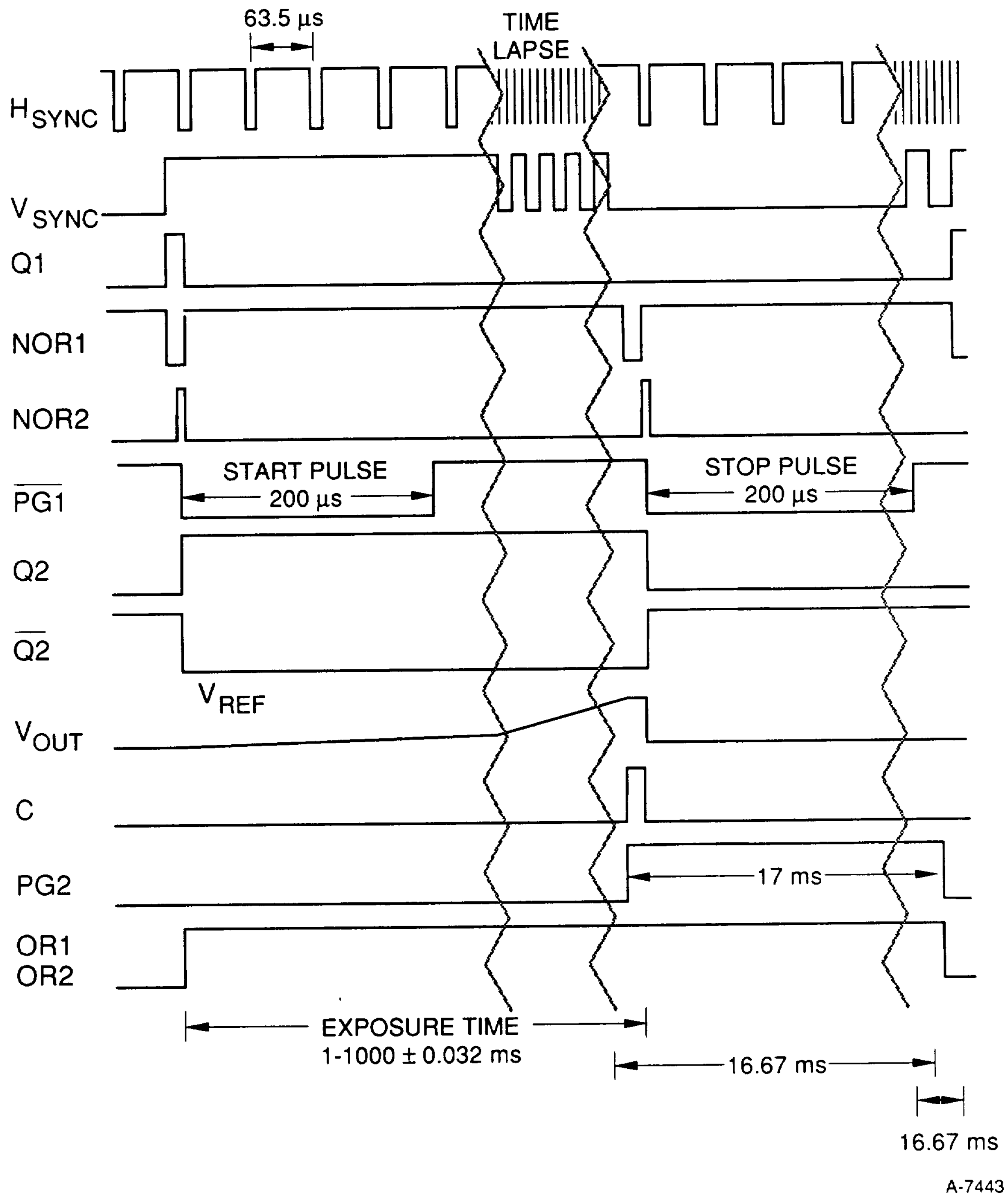


Figure D.2. - Timing diagram of camera/computer interface.

video fields. In this initial or reset state, Q1 and C are both low, so that NOR1 is high and NOR2 is low. Q2 and PG2 are low, so that OR2 and OR3 are low and flip-flop 1 is configured to toggle when the vertical sync undergoes a transition from low to high. After toggling, Q1 goes high and NOR1 goes low, so that the next horizontal sync pulse passes through NOR2. The leading edge of this pulse triggers an output pulse from $\underline{\text{PG1}}$, the exposure clear pulse.

The output pulse also toggles flip-flop 2 so that Q2, OR2 and OR3 go high, clearing flip-flop 1 to its original state and preventing it from toggling until OR3 returns to the low state. NOR1 and NOR2 return to their original states, blocking subsequent horizontal sync pulses. Also, $\overline{Q2}$ goes low, allowing the timer to start. The timer, illustrated in Figure D.3, is basically an RC circuit, amplified by the op-amp shown. It is reset by shorting the capacitor to ground through the transistor. When $\overline{Q2}$ goes low, the transistor stops conducting and, since the RC time constant has been selected to be long compared with the maximum allowable exposure time of 1000 ms, the voltage across the capacitor increases linearly with time. When the output of the op-amp reaches the reference voltage, the comparitor output C toggles from low to high. Thus, NOR1 again goes low, NOR2 transmits the next horizontal sync generating the output read pulse, toggling flip-flop 2 back to its original state, and resetting the timer. A 17 ms pulse is also generated at PG2 which holds OR2 and OR3 high for enough time to prevent flip-flop 1 from toggling during the video readout field initiated by the read pulse. At the conclusion of this period, the interface is in its reset condition, ready to start a new cycle. The output of flip-flop 2 is also used to provide a trigger pulse to

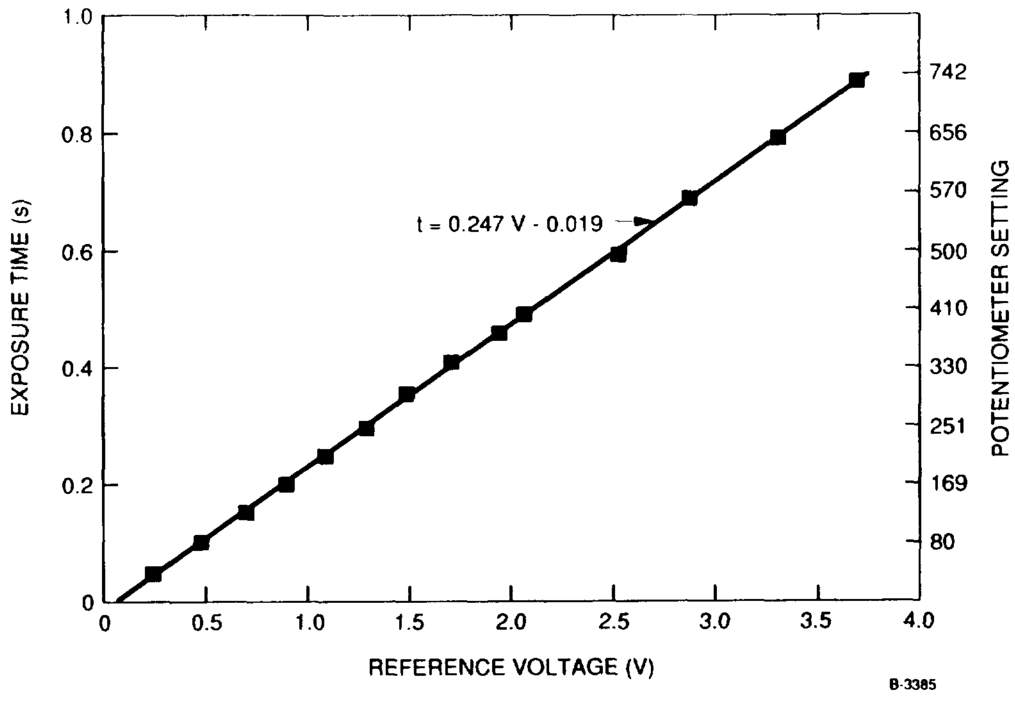


Figure D.3. - Calibration curve of exposure time vs reference voltage. Settings of the internal potentiometer which can be used to supply the reference voltage are also shown.

the DT-2861 frame grabber (via the computer sync output port on the front of the interface box), beginning digitization of the video frame containing the useful data.

Upon initial start-up of the camera and interface, the camera supplies blank fields to the video receiver which are not synchronized with the interface. After the first high-to-low transition of the vertical sync, the interface supplies a clear pulse. A read pulse is provided one user-selected exposure period later. This first read pulse is not necessarily synchronized with the camera's vertical sync, but forces the start of a new field, thereby forcing the synchronization. Since the camera is driven by the same horizontal clock as the interface, all subsequent readouts are also properly synchronized, thereby preventing garbling of the picture. Of course, if the exposure time is altered, one output field is required to re-establish this synchronization.

This circuit is completely contained within the pyrometer's interface box. The horizontal sync pulses are generated by a 15,750 Hz clock based on a 555 timing chip and are available to the user at the horizontal sync output on the front of the interface box. Vertical sync pulses are generated by dividing the horizontal syncs by 262 (available at the clock output on the interface box). In addition, a reference voltage for exposure timing can be derived from: 1) an external input attached to the "External Control" port on the front of the interface box and selecting "External" on the mode select switch; 2) the D/A output of the DT2801A board within the computer, which is internally connected when the mode select switch on the interface box is set to "Computer"; or 3) a potentiometer supplied with the interface box which is operable when the mode switch is set to "Internal". Figure D.3 is a plot of the exposure time as a function of the reference voltage.

The user will find that this interface works as described for exposure times shorter than approximately 400 ms. At longer times, noise in the timer tends to randomly change the effective exposure time by a sufficient amount to inhibit good synchronization. This limitation does not significantly impact the pyrometer operation, since changing the exposure time from 400 ms to 1000 ms (the latter being the maximum allowable by the camera) would add little additional temperature measurement capability.

The manual reset button on the front of the interface box is used to properly configure the interface's flip-flops and gates upon startup. The reset function is provided by a bounceless switch which toggles from low to high when pressed and held. This low signal is connected directly to the flip-flop 2 clear input, and to flip-flop 1's clear through OR3. Clear and read pulses to the camera are suppressed when the reset button is held down. Holding the button for one second also resets the camera so that the next input pulse is interpreted as a read. This function should be used if, after changing the integration time or as the result of an electrical spike, the camera interprets clear and read pulses backwards as indicated by excessively bright video images.

APPENDIX E

Calibration

As discussed in Appendix A, the output of the pyrometer is related to the temperature of the observed object via the Planck equation and a number of parameters that are determined by the optical and electronic characteristics of the system. These parameters are independent of temperature and, for each wavelength, may be combined as in Eq. (A.7) into a single constant B_λ that, when multiplied by $F_\lambda(T) = 1/[\lambda^5(\exp(C_2/\lambda T) -1]$, yields the output signal, $V_\lambda(T)$. The six values of B_λ are called the pyrometer's calibration constants. Though they can, in principle, be calculated from first principles, the calibration constants are determined most precisely by measurements. The procedures that were used to calibrate the multicolor imaging pyrometer, and the results, are described here.

Calibration Procedure

The multicolor imaging pyrometer was calibrated using two blackbody sources: An Infrared Industries Model 413 blackbody capable of achieving a maximum temperature of 1273 K (1000 C), and a Thermogage Model 24 kW source, capable of exceeding 2500 K. Each source is provided with an NBS traceable temperature calibration. The general calibration procedure was to place the blackbody's aperture (or a unity magnification an image of it) at the pyrometer's object plane, adjust the aperture diameter to about 2.5 mm, and then measure the temporally and spatially averaged output signal (usually in terms of gray levels digitized by the frame grabber) from each of the pyrometer's images as a function of temperature. The program "MANUAL.C" is useful for measuring gray levels. The calibration constants were found by plotting $V_\lambda(T)$ against $F_\lambda(T) = [\lambda^5\{\exp(C_2/\lambda T) - 1\}]$ and calculating the slope of the resulting line. This information is used in operating the pyrometer to calculate look-up tables that relate each gray level in each image to a corresponding temperature.

Results

The calibration curves for each of the six colors are shown in Figures E.1 through E.6. The linearity of the response is clearly excellent throughout the temperature ranges over which each color is designed to operate. It should be noted, though, that the calibration lines do not pass through the origin - each has a slight offset. At the four longer wavelengths, the offset is negative, a result of the CCD camera's non-linear response at low gray levels discussed in Appendix B. At the two shorter wavlengths the offset is positive, which we believe is caused by a small amount of scattered light leaking into the region of interest on the CCD. Since the pyrometer is not meant to provide accurate data at any of the wavelengths when using gray levels below 20, these offset values, O_λ, are easily accomodated by including them in the calibration, viz:

$$V_\lambda(T) = B_\lambda(T)F_\lambda(T) + O_\lambda \quad .$$

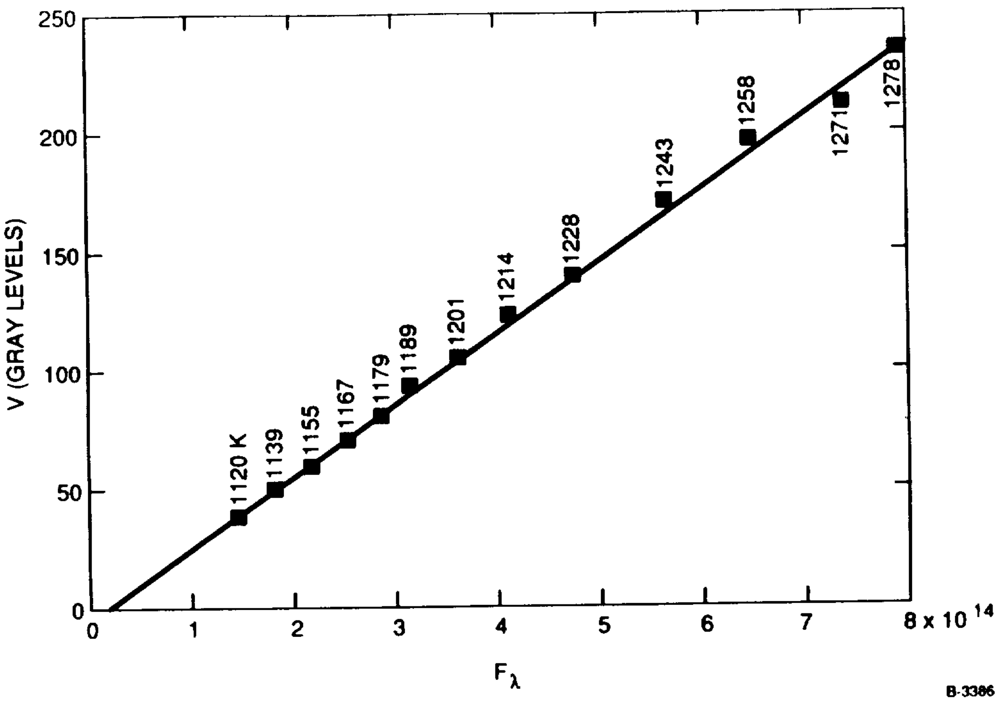


Figure E.1. - Calibration curve for λ = 931.4 nm. Least square fit yields $V = 3.049 \times 10^{-13} F_\lambda - 6.044$

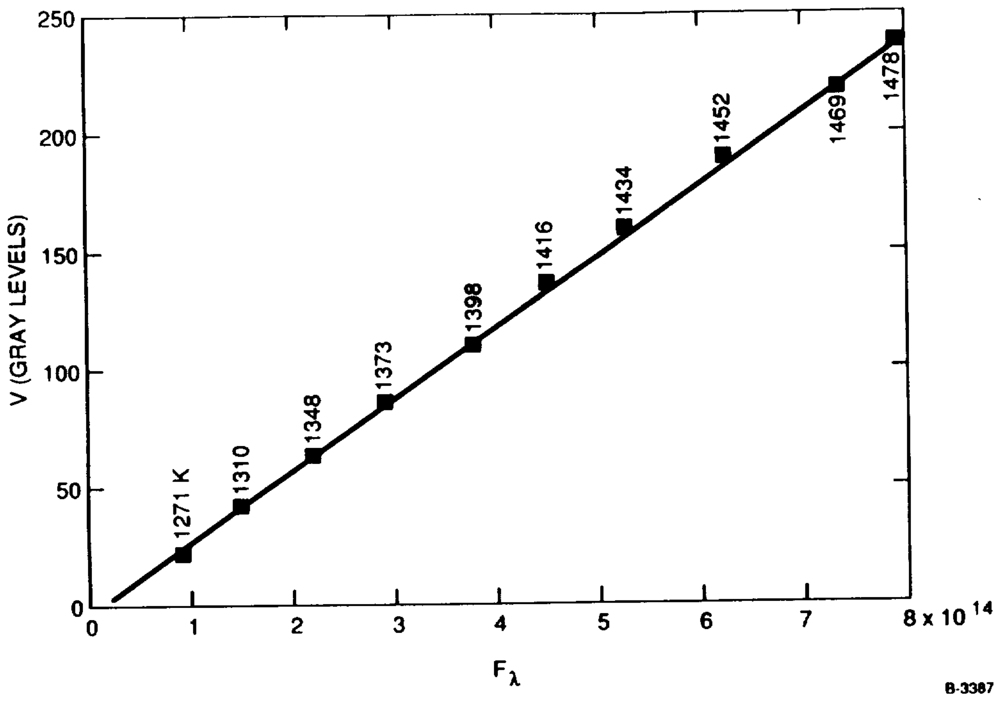


Figure E.2. - Calibration curve for λ = 733.0 nm. Least square fit yields $V = 3.095 \times 10^{-13} F_\lambda - 4.461$.

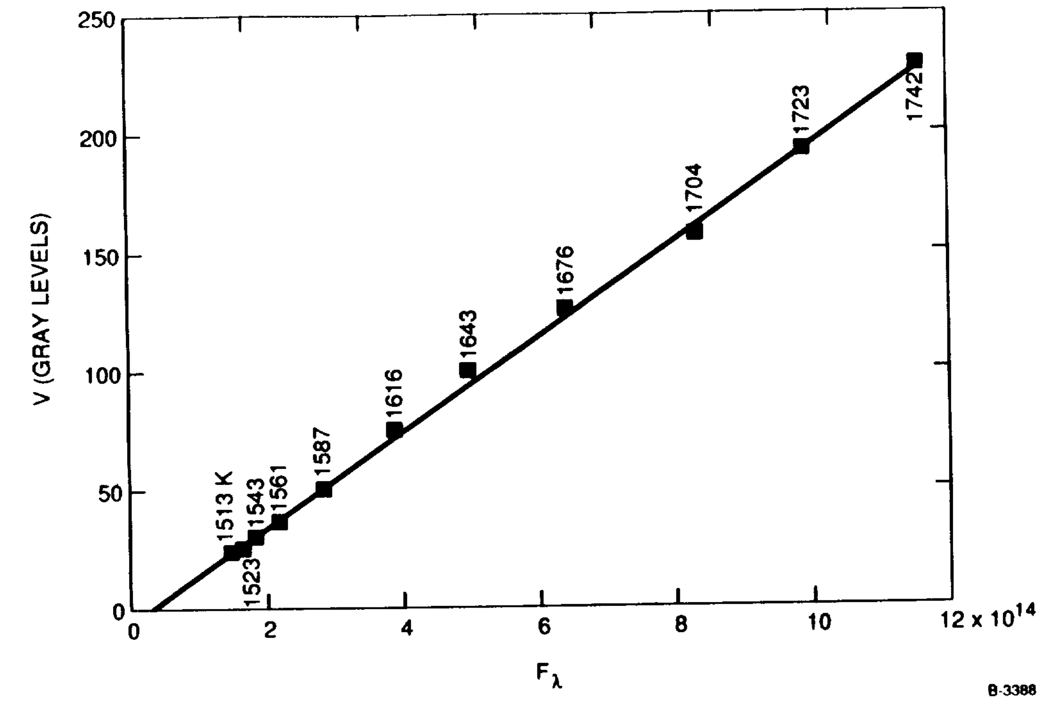


Figure E.3. - Calibration curve for λ = 590.0 nm. Least square fit yields $V = 2.034 \times 10^{-13} F_\lambda - 7.588$.

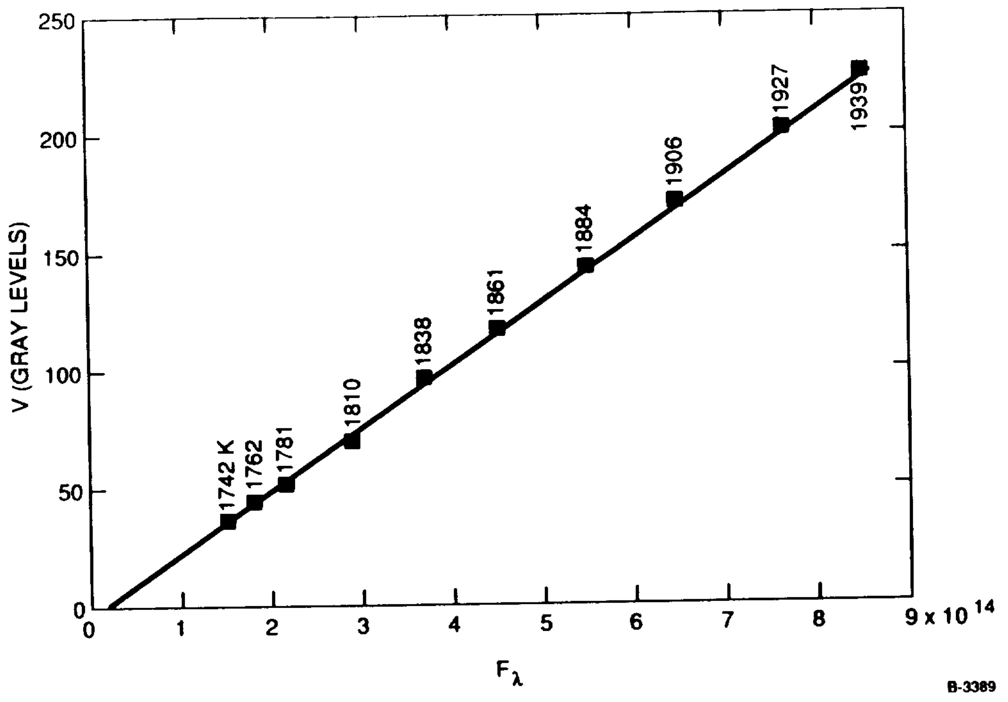


Figure E.4. - Calibration curve for λ = 486.3 nm. Least square fit yields $V = 2.705 \times 10^{-13} F_\lambda - 6.91$.

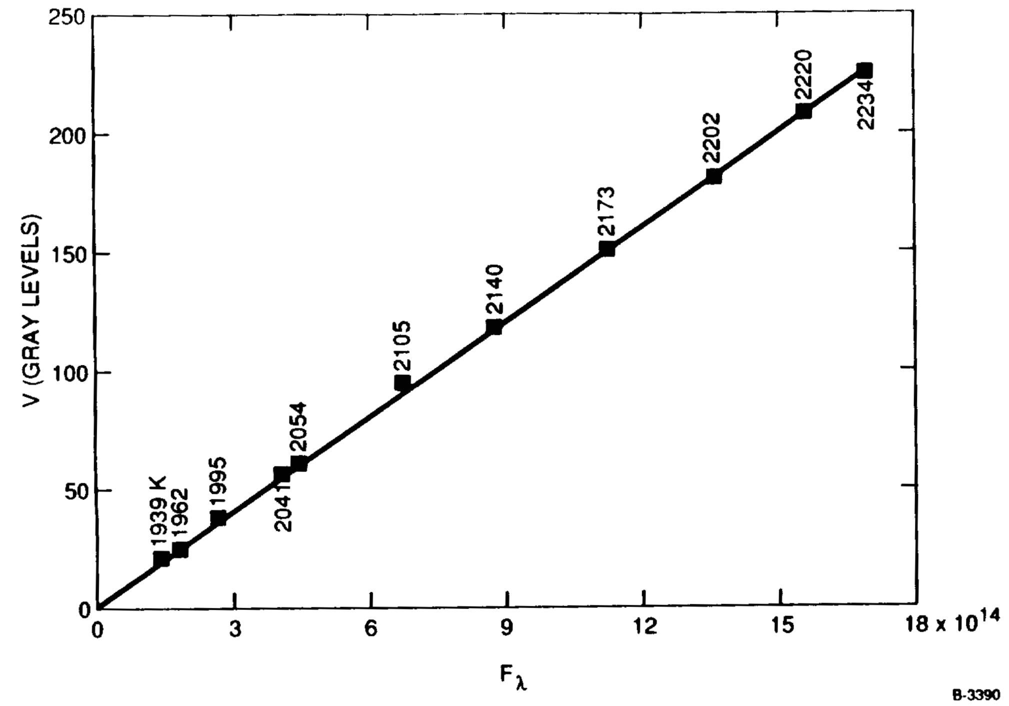


Figure E.5. - Calibration curve for λ = 421.5 nm. Least square fit yields $V = 1.333 \times 10^{-13} F_\lambda - 0.541$.

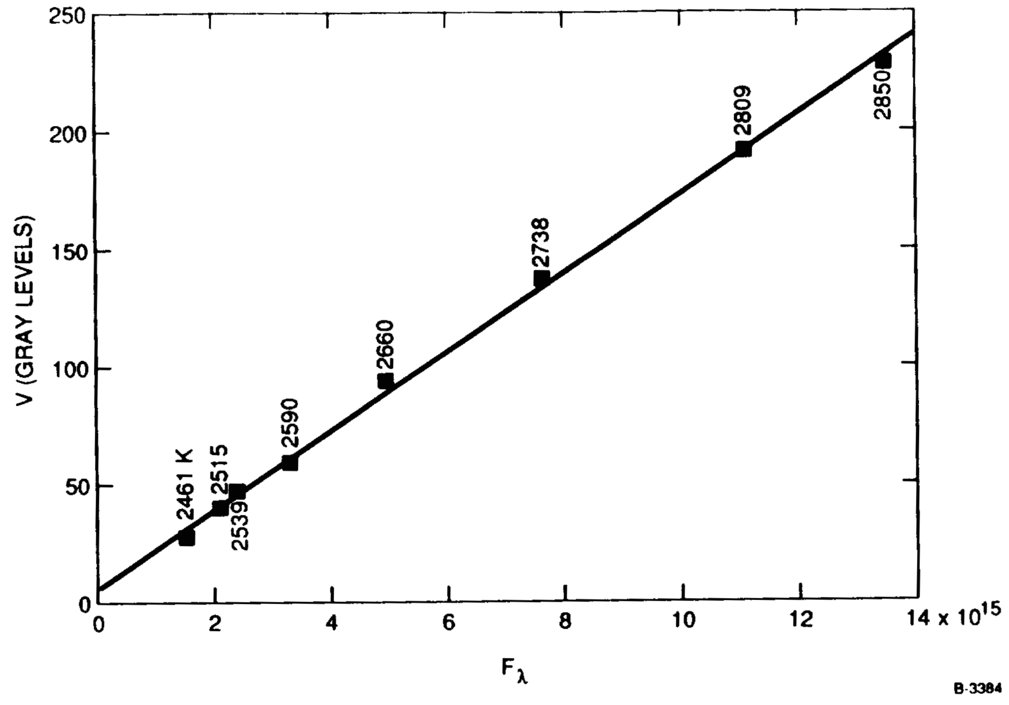


Figure E.6. - Calibration curve for λ = 360.5 nm. Least square fit yields $V = 1.681 \times 10^{-14} F_\lambda - 8.029$.

The values for B_λ and O_λ for the six colors are provided in Table E.1. All calibrations were performed at the standard integration time of 16.67 ms, except for the longest wavelength (931 nm) which used a 50 ms exposure. Knowledge of the calibration exposure time is included in the data analysis programs, and any recalibrations should be performed using the same periods.

TABLE E.1. CALIBRATION CONSTANTS

Wavelength (nm)	B_λ (gray levels/cm^5)	O_λ (gray levels)
931.4*	3.049×10^{-13}*	-6.04
733.0	3.095×10^{-13}	-4.46
590.0	2.034×10^{-13}	-7.59
486.3	2.705×10^{-13}	-6.91
421.5	1.333×10^{-13}	+0.54
360.5	1.681×10^{-14}	8.03

*50 ms exposure, compensated within program

APPENDIX F

Pyrometer Software

Three computer programs, QUICKIE, MANUAL, and PYRO have been supplied with the pyrometer. QUICKIE and MANUAL together provide for high speed data acquistion and detailed post-test interactive analysis, while PYRO is designed for use as an automatic temperature measurement and control routine. Details of the programs, along with program listings, are provided here. Since QUICKIE and MANUAL are subprograms of PYRO, the latter is discussed first.

F.1 Algorithm and structure of PYRO.C

PYRO.C is the source code for PYRO.EXE, the program that is executed by typing PYRO after the DOS prompt. The function of PYRO has been discussed previously in Subsection 6.2.

Program flow. - After starting PYRO, the program first continually acquires and displays video frames while prompting the user for operating parameters, i.e., the desired integration time and assumed object emissivity. It then sets the voltage output from the D/A converter on the DT2801A board to control the integration time as discussed in Appendix D, calculates temperature lookup tables and false color palettes, loads the palettes onto the frame grabber, and then prompts the user to cover the pyrometer aperture so that the dark signal can be acquired. Upon receiving the trigger signal, it acquires ten dark frames and then computes an average dark signal for each pixel. This dark frame is stored in one of the sixteen frame buffers on board the DT2861 frame grabber.

After prompting for and receiving another user signal, the program begins pyrometry by acquiring the first video frame provided by the camera. If operating in the variable exposure mode, frames are acquired only upon receipt by the frame grabber of a trigger signal supplied by the exposure control electronics via the "Computer Sync" line. The background frame is subtracted from the acquired frame during the digitization process.

The program then searches for the image at the first (i.e., longest) wavelength. If no image is found, the program displays a message and goes back for another frame. If an image is found, the program determines the (x,y) coordinates of the image centerpoint as well as the x and y values at its outer boundaries. The program then finds the first image that contains useful temperature information. To this end, it steps through the images from each wavelength (which are in deterministic locations relative to the image from the first wavelength) and selects the longest wavelength image that is not saturated. The corresponding false-color palette is also selected.

The program next calculates a "most prevalent" temperature of the object surface by finding the peak value a histogram of the pixel grey levels in the selected image. Using the previously calculated lookup tables, the most prevalent gray level value is converted to a temperature. That temperature is displayed on the computer monitor and supplied to the CONTROL subroutine, which can be written by the user to control the heating process. The program finally checks to see if the user has signalled to end the program and, if not, then goes back for another frame.

Details of Methodology. - The various routines used in PYRO.C are described here to aid the user in interpreting the source code. A listing of the code is provided at the end of this appendix.

a) Setting the integration time

To set the integration time (the variable "tau" in the code), a voltage is output from the D/A converter on-board the DT2801A board and, if the electronics box (see Appendix D) is set in the "Computer" mode, that voltage is supplied directly to the comparator in the integration time control circuitry. The voltage is set to agree as closely as possible with the calibration curve of Figure D.3. However, since a digital to analog board is used, only discrete voltage levels are possible. The program automatically adjusts the requested integration time to the closest possible setting less than or equal to that requested.

b) Acquiring a frame

Frame acquisition on the DT2861 board is performed using the DT-IRIS subroutine IS_ALU_ACQUIRE_EXTERNAL (ISAAQE). This single command instructs the computer to wait for the Computer Sync pulse from the integration control electronics, acquire a frame, and subtract the baseline dark signal. After processing the acquired data, the frame is continually displayed in false color on the video monitor until acquisition and processing of the next frame is complete.

Note that during development of the pyrometer, a DT2851 frame grabber was used. This board did not have the capability for external triggering or on-board background subtraction. Therefore, one of the analog inputs on the DT2801A board was used to continually check the Computer Sync line for receipt of the new frame signal. Sections of the delivered code contain comment lines which are the commands for running the DT2851. These lines are used at PSI for continued code development.

c) Building the temperature table and false color palettes

In the load-palette subroutine, six temperature tables, one value per grey level for each color, are formed using the Planck equation, the filter wavelength, assumed object emissivity input at program startup, integration time, and the calibration tables read in from the file CALIB.DAT. Then a false color palette is created for each filter. Each false color palette comprises three 8-bit output look-up tables (OLTs) that assign - to the red, green, and blue, outputs of the frame grabber - intensity levels corresponding to each of the 256 digitizer gray levels. Thus, each input gray level can be mapped into any one of 2^{24} available colors. The palettes are created so that small temperature gradients are displayed as dramatic differences in color on the video monitor, using a procedure described below.

As discussed in Appendix C, the temperature ranges measured by the six colors have some overlap. The temperatures measured by the upper range of one wavelength can be detected, albeit with less precision, by the lower range of the next shorter wavelength. The palettes are constructed so that each measured temperature is displayed by a unique color, regardless of the wavelength at which it is measured. Thus, different OLTs are required for each of the six wavelengths. All six tables are calculated upon startup of the PYRO program and stored in OLT buffers on-board the DT2861 frame grabber. However, only one table can be used at any particular time. The computer is programmed to automatically select the output look-up table that corresponds to the wavelength used to calculate the control temperature.

For each gray level to be displayed as a unique color, and to visually enhance the appearance of temperature gradients, the red, green, and blue look-up tables are created from three periodic sawtooth functions, each having a different frequency:

```
X(N) = (N MOD 255/3) * 3
Y(N) = (N MOD 255/32) * 32
Z(N) = (N MOD 255/64) * 64
```

where the result of the MOD operation is the remainder of the quotient of N (used to denote the gray level) and the term to the right of MOD. These three functions execute 3, 32, and 64 cycles respectively over the 255 input gray levels. If X, Y, and Z were assigned to the colors Red, Green, and Blue (R,G,B) respectively, then at gray level 3 x 64 = 192, the colors would start to repeat. To prevent this repetition, the assignment of colors is permuted three times over the 255 gray levels. For example:

```
      N <  86:  R(N) = X(N); G(N) = Y(N); B(N) = Z(N)
 85 < N < 171:  R(N) = Z(N); G(N) = X(N); B(N) = Y(N)
170 < N < 256:  R(N) = Y(N); G(N) = Z(N); B(N) = X(N).
```

This specific set of permutations is used for the 930 nm look-up table, and is stored on disk in the file PAL1.LUT (which can be read by a text editor). Different permutations are used for the remaining wavelengths, following two steps illustrated in Figure F.1: 1) For the gray levels where the temperature overlaps that of the previous wavelength, the RGB values used in the previous wavelength to display those temperatures is used. That is, if the image of the heated object appears bright orange as its temperature approaches the upper limit of one wavelength and the active image switches to the next shorter wavelength, then the color remains bright orange; and 2) For gray levels corresponding to temperatures higher than those to which the previous wavelength is sensitive, the sawtooth pattern is again implemented, though the order of the permutations is selected to ensure continuity as the temperature increases.

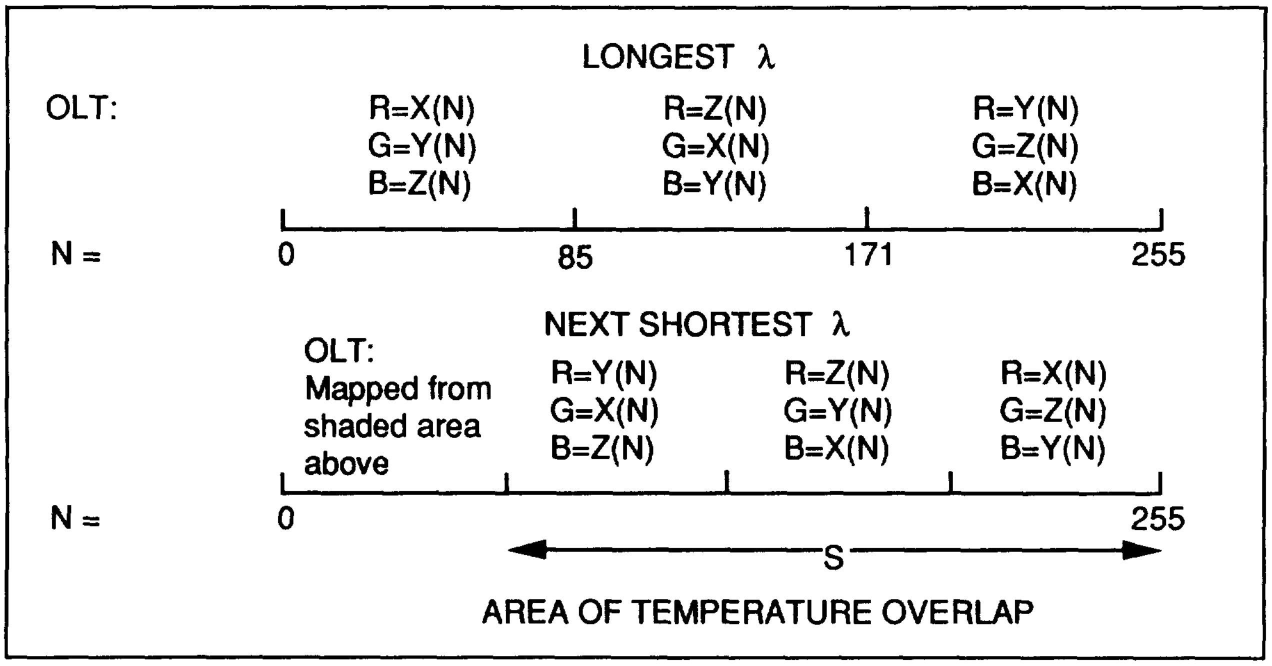


Figure F.1. - Methodology used to create output look-up tables.

d) <u>Finding the image</u>

To determine the control temperature, PYRO must automatically find the image containing the relevant data. Since, as described by Eq. (C.1), the position of any image can be deduced from the position of any other image, the process of finding the images is reduced to finding the position of the first image, which is known to be in the lower left quadrant of the screen. To locate this image, the quadrant is divided into several thin strips which are scanned to find pixels having gray levels significantly above the noise. Since the second image will always be about 128 pixels to the right of the first image, and the sixth image about 160 pixels above the first, the strips are selected to be 100 pixels long and each is scanned from left to right starting from the lower left corner and working upwards. Once several (defined by the variable NUM_ADJ_ABOVE_BGND) adjacent pixels have been found above the noise level (called "bgrnd" in the code), the image is considered detected.

The next step is to find the center of the image. This is achieved by assuming that the image is vaguely round and determining the horizontal and vertical limits of the circle. A three step process is used. First, the search procedure just described yields a left bound located somewhere near the bottom of the image. The scan of that row is continued to the right until the right edge is detected. The average of the x values provides a vertical centerline which is scanned to determine the upper and lower y bounds. The average of these values provides an x centerline, which is then scanned to find the left and right x bounds. The two x and two y bounds effectively form a rectangle that surrounds the image. (Note, though, that if the image is very non-circular this technique may not work. For strangely shaped objects another routine should be developed. Also, if the first image is outside the allowable 1 cm diameter field of view, the algorithm may conclude that another image, i.e., the second or sixth, appears to be the first, or it may find no image at all.) Once the center of first image is located, all the other five center-points are positioned relative to it according to the relationships in Appendix C.

After locating the images, the longest wavelength image is tested to determine if any pixels are saturated. If so, the next shortest wavelength image is similarly tested, and so on, until an image with no saturation is found, or all images are determined to have saturation. The first filter without saturation, if any, is selected for use in determining the typical temperature of the surface.

e) Determining the typical temperature

A histogram of the pixel values of the selected image is calculated. For this purpose, the rectangle formed by the x and y bounds of the image is taken to be the area covered by the image. Although some unilluminated pixels may be included in this area, their number is typically insufficient to affect the position of the histogram peak, unless steep temperature gradients are present. To be safe, pixel values below the noise level are not considered in the histogram. The temperature corresponding to the pixel value at the peak of the histogram is taken to be the typical temperature of the object surface. It is displayed on the computer monitor along with the gray level and the number of the filter for the image used, and is supplied to the CONTROL subroutine.

Using PYRO as a temperature controller. - The DT2801A analog/digital conversion board provides two independent analog voltages outputs and sixteen (two bytes) digital outputs. One of the analog outputs is used to control the integration time, but any of the remaining ports can be used to control external devices based on the temperature calculated by the program. PCLAB subroutines can be used to access these ports. It is recommended that the DT2801 and PCLAB documentation provided with the pyrometer be consulted for details.

The PYRO program has been written so that the measured temperature is supplied as an INTEGER*2 variable to an external subroutine called CONTROL. As delivered, CONTROL is simply a dummy routine that returns to PYRO without taking any action. This feature is provided so that the user can write a CONTROL subroutine using the PCLAB library functions callable from Basic, Fortran, C or Pascal, compile that subroutine, and link it with PYRO.OBJ, thereby controlling the heating process as desired without having to modify PYRO.C.

F.2 Algorithm and structure of QUICKIE.C

QUICKIE.C is a subset of PYRO.C to be used primarily for frame acquisition and storage only. QUICKIE acquires and displays frames as fast as is allowed by the exposure control electronics - up to 60 fields per second in the standard mode of operation. Since QUICKIE does no analysis, unlike PYRO there is no computation time consumed between frame acquisitions. Applications of QUICKIE have been suggested in Subsection 6.2.

Program flow. - The startup routine in QUICKIE is similar to that of PYRO - the integration time is requested and set, and an option to perform background subtraction or not is presented. If requested, ten dark frames are acquired, averaged, and stored as in PYRO. After waiting for the user prompt, the program begins frame acquisition. If in variable integration time mode, after background subtraction (if selected) each acquired field is displayed on the video monitor until the next field is digitized. This procedure prevents the blank fields supplied by the Sierra Scientific camera (see Appendix D and the camera manual) from creating disturbing flicker. Frames are continually displayed until the user signals a stop by pressing the END key.

F.3 Algorithm and structure of the MANUAL.C

MANUAL.C is partially a subset of PYRO.C, but includes an interactive image analysis routine as described in Subsection 6.2. MANUAL can be used to acquire and interactively analyze single frames from the imaging pyrometer, or it can be used to process images gathered and recorded earlier with QUICKIE. Like PYRO, MANUAL calculates and displays the most prevalent temperature of the object surface. In addition, to enable measurement of temperature gradient information that may be spread over two or more of the six images, it allows the user to measure temperatures point by point in each image, and to hop from one of the images to the corresponding position on any other.

Program flow. - MANUAL starts like PYRO and QUICKIE with dark frame acquisition, and requests for operating parameters. However, at the user prompt to start data acquisition, MANUAL acquires only a single frame of data and, if requested, subtracts the dark signal. The program then performs the same data analysis as PYRO, and presents the deduced temperature on the computer monitor along with selecting a false color palette based on which image is unsaturated.

At the conclusion of the analysis, the routine deviates from that of PYRO. A crosshair cursor is placed on the pixel at the center of the chosen image, and the gray level of that pixel is converted to a temperature and displayed. The user is allowed to move the cursor around that image or to other images using the keyboard. The arrow keys move the cursor around within an image, while the +, -, TAB, BACKTAB (shift-TAB) keys enable movement between images. The + and - keys move the cursor to the center of the next (shorter wavelength) or previous (longer wavelength) image, while the TAB and shift-TAB keys move to the point in the next or previous image corresponding to the cursor location in the current image. These features determine the relative positions of points in different frames using Eq. (C.1). The HOME key moves the cursor to the center of the current image. A new frame is captured by pressing ENTER.

Methodology. - The routines used in MANUAL are the same as those for PYRO, except in the interactive analysis which employs the DT-IRIS cursor positioning subroutines IS_SET_CURSOR_POSITION and such. The temperature at each selected pixel is read from the appropriate table calculated at program startup.

PROGRAM LISTINGS

PRECEDING PAGE BLANK NOT FILMED

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "\iris\iserrs.h"
#include "\iris\isdefs.h"

#define aaa -5.121e-06
#define bbb 0.4914
#define ccc -2.7572
#define fntemp(y) ((-1*bbb+sqrt(bbb*bbb-4.*aaa*(ccc-y*1000./25.)))/(2.*aaa)+273.)
#define SYNC_EXT 1
#define cls printf("")
#define NUM_ADJ_ABOVE_BGND 5
#define MIN_BGRND   3
#define MAX_BGRND   25
#define HOME (0x4700)
#define END (0x4f00)

double   cslope[6],offset[6];

int  idv(p,m,v)
int  p,m;
unsigned int  *v;

{
double  a,b,c;
int i,j,k;
a=2.;c=3.; /* dummy vars*/

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     if ((inp(0x2ed)&1)!=1)  {
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2)
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ed,(char) 6);
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2)
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(char) p);
     }

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while ((inp(0x2ed)&1)!=1)
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     *v = (inp(0x2ec)&m); /* read the data byte */
```

```
}

void xmv(chan,vlts)
int  chan;
double    *vlts;

{
     /* Put this in when thermocouple measurements are desired. */
}

void xdv(chan,vlts)
int chan;
int vlts[];

{
double  a,b,c;
int i,j,k;
a=2.;c=3.; /* dummy vars*/

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     if ((inp(0x2ed)&1)==1){
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          while ((inp(0x2ed)&1)!=1)
               for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          j = (inp(0x2ec)); /* read the data byte */
     }
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ed,8); /* command code for output-analog */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) chan); /* channel select, 0 or 1, or -1 for both */
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) vlts[0]%256); /* lo byte of voltage */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) vlts[0]/256); /* hi byte of voltage */
     if (chan == -1) { /* output bytes for second channel */
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          while((inp(0x2ed)&2)==2) /* wait until board ready */
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(char) vlts[1]%256); /* lo byte of voltage */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2) /* wait until board ready */
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(char) vlts[1]/256); /* hi byte of voltage */
     }
}

void set_itime(tau)
double    tau;
#define   lv (-10.)
{
double    v,actualv,cpv;
int       analog;
     cpv = 4096/20;
     v = (tau + 19.) / 247.;
     if (v > 5.)    v = 5.;
     if (v < .076)  v = .076;
     analog = (v - lv) * cpv;
     actualv = analog / cpv + lv;
     xdv(0,&analog);
     printf(" Integration time = %lg (ms)",actualv * 253.53 - 16.61);
}

void load_palette(c2,l,temp,eps,tau) /*** create palettes and temperatures table ***/
double    c2,l[6],temp[6][256],eps,tau;

{
FILE      *pf,*tf;
double    t,vo,factor;
int       pl,index,ado,adi,red[2][256],green[2][256],blue[2][256],flag ;

     flag = 0;
     red[0][0]=green[0][0]=blue[0][0]=0;
     red[1][0]=green[1][0]=blue[1][0]=0;
     tf = fopen("temp.dat","w");
     temp[0][0] = 0.;
     pf = fopen("pal1.lut","r");
     for (index=1;index<=255;index++) {
          fscanf(pf,"%d%d%d",&(red[0][index]),&(green[0][index]),
           &(blue[0][index]));
          factor = cslope[0]*eps*tau/
            (50.*l[0]*l[0]*l[0]*l[0]*l[0]*((double)index-offset[0]))+1;
          if (factor<=0)
               flag = index;
          else
               temp[0][index] = c2 / (l[0]*log(factor));
          fprintf(tf,"%3d   %6.1f\n",index,temp[0][index]);
     }
```

```
    fclose(pf);
    for (index=0;index<flag;index++) /* in case of positive calibration offset */
         temp[0][index] = temp[0][flag+1];
    ISLDOT(0,&(red[0][0]),&(green[0][0]),&(blue[0][0]));
    printf(" palette 0\n");
    for (pl=1;pl<6;pl++) {
         flag = 0;
         adi = 1;
         temp[pl][0] = 0.;
         for (ado=1;ado<=255;ado++) {
              vo = (double)ado - offset[pl];
              factor = cslope[pl]*eps*tau/
               (16.67*l[pl]*l[pl]*l[pl]*l[pl]*l[pl]*vo) + 1;
              if (factor>0)
                   temp[pl][ado] = c2/(l[pl]*log(factor));
              else {
                   flag = ado;
                   red[pl&1][ado]=green[pl&1][ado]=blue[pl&1][ado]=0;/*black*/
                   break;
              }
              for (t=temp[pl][ado]; t>temp[pl-1][adi] && adi < 256; adi++) ;
              if (adi==256) {
                   for (index=ado;index<=255;index++) {
                        red[pl&1][index] = (red[pl&1][index-1] + 3) % 256;
                        green[pl&1][index] = (green[pl&1][index-1] + 32) % 256;

                        blue[pl&1][index] = (blue[pl&1][index-1] + 64) % 256;
                        temp[pl][index] = c2/(l[pl]*log(cslope[pl]*eps*tau/
                          (16.67*l[pl]*l[pl]*l[pl]*l[pl]*l[pl]
                          *((double)index-offset[pl])) + 1));
                   }
                   goto done_pal;
              }
              red[pl&1][ado] = red[!(pl&1)][adi];
              green[pl&1][ado] = green[!(pl&1)][adi];
              blue[pl&1][ado] = blue[!(pl&1)][adi];
         }
done_pal:
         for (index=0;index<flag;index++) /* in case of positive calibration offset */
              temp[pl][index] = temp[pl][flag+1];
         ISLDOT(pl,&(red[pl&1][0]),&(green[pl&1][0]),&(blue[pl&1][0]) );
         printf(" palette %1.1d\n",pl);
    }
    fclose(tf);
}

/*********** color_time -- print color bar at bottom of video **************/

void      color_time(fm,temp,pl)
int  fm;
double temp[6][256];
```

```
int pl;

{
int       gpx,gpy,j,idx[256],cn,ct,rn,temparr[10];
char tempstr[10];

     gpy=440;
     gpx=400;
     for (j=0;j<=255;j++)
          idx[j]=j;
     cn=128;
     ct=256;
     for (rn=440;rn<=470;rn++)
          ISPUTP(fm,rn,cn,ct,idx);
     sprintf(tempstr,"%6.1lf",temp[pl][1]);
     IS_SET_GRAPHIC_POSITION (430,128);
     for (j=0;temparr[j]=tempstr[j];j++);
     ISTEXT(fm,j-1,temparr);
     sprintf(tempstr,"%6.1lf",temp[pl][255]);
     IS_SET_GRAPHIC_POSITION (430,384);
     for (j=0;temparr[j]=tempstr[j];j++);
     ISTEXT(fm,j-1,temparr);
}

/************** mprint -- print text on the video monitor *****************/

void mprint(fm,x,y,str)
int  fm,x,y;
char *str;

{
int  chr[21];
int  ct,h;
     ct = strlen(str);
     for (h=0;h<ct;h++)
          chr[h] = str[h]; /* convert to int array (16 bit) from char array (8 bit) */
     ISGPOS(y,x);
     ISTEXT(fm,ct,chr);
}
/****************************************************************************/
/*************************** pyro.c main ************************************/
/****************************************************************************/

/* labels: errexit,finalexit */

main (argc,argv)

int argc;
char *argv[];

{
```

```
int  hist[256], tm[8], pix[512], pixa[512], pixb[512], vpix[2], bright[100],
     back[100], bright1[2], bright2[2], back1[2], back2[2], dif1[100],
     dif2[100], fm, value, i, j, k, n, z, r, image, time4, avcnt, bgrnd,
     status, time5, inport, mask, record, gain, chan, x1, y1, x2, y2, dx, dy,
     sx, sy, row, col, clr, centx, centy, nm, ra, ca, histmax, hmax,
     frmav, sat, rs, cs, dif, chr[30], pal[6][3][256], x1a, x1b, rowmin,
     stdvid, satlevel, ans;

long      tctemp[3], volts[3],bhist[256];
FILE      *f1;
float     vits;
double    a,b[6],l[6],c,c2,temp[6][256],calib,eps,tau;
time_t    time1;
struct    tm   *time2;
char      time3[26], imagestr[30], tempstr[30];

     cls; /* clear computer monitor */
     c2=1.4388;
     value=1;
     image=0;
     status = ISINIT(); /** initialize frame grabber and driver and DT-IRIS **/
     if (status>0)
          goto finalexit;
     f1 = fopen("calib.dat","r");
     for (z=0; z<6; z++) {
          fscanf(f1,"%lg%lg",&(cslope[z]),&(offset[z]));
          printf ("calib: cslope %lg offset %lg\n",cslope[z],offset[z]);
     }
     fclose(f1);
     l[0]=0.00009314;
     l[1]=0.00007330;
     l[2]=0.00005900;
     l[3]=0.00004863;
     l[4]=0.00004215;
     l[5]=0.00003605;
     ISSYNC(SYNC_EXT); /* set sync to external */
     ISDISP(1); /* turn display on */
     IS_SELECT_OUTPUT_FRAME(1); /* select frame buffer one for output (display) */
     IS_SELECT_INPUT_FRAME(0); /* select frame buffer zero for input */
     fm = 0;
/*     cls;*/
     printf(" Enter desired integration time (ms), zero for standard video:       ");
     while (!scanf("%lf",&tau))
          printf("= 0);
     if (stdvid)
          tau = 16.67;
     else
          set_itime(tau); /* select closest possible integration time to that
                                    requested */
     bgrnd=(stdvid)?12:7; /* Pick a level to consider as background even after dark frame
is subtracted.  This is modified later as nature of image becomes apparent. */
```

```
     ans=0;
     do {
          if (stdvid) { /* standard video rate */
               ISACQ(fm,1);
          }
          else { /* variable int time */
               ISACQE(fm,1);
               ISWAQC();
 /* this code needed for a DT-2851 frame grabber: */
               /*ISINFR(fm);*/
               /*ISPASS(); *//* pass-thru mode on frame grabber until camera has finished
integrating */
               /*do idv(0,1,&value); while(value==0);*/ /* loop until signal that camera
*/
               /*do idv(0,1,&value); while(value==1); *//* is done integrating */
               /*ISFREZ();*/ /* freeze frame */
          }
          /* color_time(fm,temp,clr); /* color bar and time display */
          IS_SELECT_OUTPUT_FRAME(fm);
          fm = 1 - fm; /* toggle frame betw. 0 and 1 */
          IS_SELECT_INPUT_FRAME(fm);
          printf (" Press HOME key to start; END to exit. ");
          if (kbhit()) {
               ans=getch();
               if (ans==0) /* extended key code */
                    ans=(getch())<<8; /* get 2nd byte and put in upper half of word */
          }
     } while (ans !=HOME && ans!=END);
     if (ans==END) {
          ISEND();
          return;
     }
     cls;
     printf(" Enter emissivity:      ");
     scanf("%lf",&eps);
     cls;
     while (kbhit()) getch(); /* clear key buffer */
     printf(" Prepare dark frame.  Press any key when ready. ");
     getch();
     cls;
     avcnt = 10;
     ISDISP(0);
     ISFCLR(0);
     ISFCLR(1);
     ISOTFR(0);

     for (k=0;k<=255;k++) {
          pix[k]=k;
          pix[k+256]=255;
     }
     ISLRLT(0,pix); /* load RLUT -- Resultant Look-up Table */
```

```
   for (frmav=0;frmav<avcnt;frmav++) {
        if (stdvid) { /* standard video rate */
             ISALAQ(25,1,0,1);
        }
        else {  /* variable int time */
             ISAAQE(25,1,0,1);
             ISWAQC();
        }
   }
   ISDIVC(1,avcnt,2); /* divide buffer by avcnt to find average backgrnd */
   /*** do histogram of background ***/
   if (stdvid)
        ISHIST(2,bhist);
   else {
        for (k=0;k<=255;k++)
             bhist[k]=0;
        for (row=2;row<=454;row+=2) {
             ISGETP(2,row,0,512,pix);
             for (j=0;j<512;j++) {
                  bhist[pix[j]]++;
             }
        }
   }
   for (j=0;j<=255;j++)
        if (bhist[j]>50)
             satlevel=j; /* pick highest light level significantly present
                                 in the background */
   satlevel=255-satlevel; /* subtract this from 255 to get the saturation
                                  light level */
printf ("Saturation level = %d\n",satlevel);

   while (kbhit()) getch(); /* clear key buffer */
   printf(" Ready for measurement.  Press any key to start. ");
   getch();
   cls;
   printf ("Loading false-color palettes...\n");
   load_palette(c2,l,temp,eps,tau);
   printf ("...done loading palettes.\n");

   for (k=0;k<=255;k++) {
        pix[k]=0;
        pix[k+256]=k;
   }
   ISLRLT(0,pix); /* load RLUT -- Resultant Look-up Table */


   ISDISP(1);
   while (1) {
        clr = 0;
        record = 0;
        sat = 0;
```

```
fm = 1 - fm;
  /* ISINFR(fm); not needed; alu_acquire does this */
printf(" Acquiring... ");
if (stdvid) { /* standard video rate */
     ISALAQ(38,2,0,fm);
}
else { /* variable int time */
     ISAAQE(38,2,0,fm);
     ISWAQC();
}
/* cls; */
/********* find image **********/
n = 100+NUM_ADJ_ABOVE_BGND;
y1=y2=x1=x2=0;
for (cs=2;x2==0 && cs<=200;cs+=100) {
     for (r=454; !x2 && r>=154; r-=2) {
          j = 0;
          ISGETP(fm,r,cs,n,pix);
          while (x1==0 && j<n-NUM_ADJ_ABOVE_BGND) {
               z = 0;
               j++;
               while (pix[j]>bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                    z++;
                    j++;
               }
               if (z==NUM_ADJ_ABOVE_BGND)
                    x1=j-NUM_ADJ_ABOVE_BGND+cs;
          }
          if (x1) {
               n = 256 - x1 + NUM_ADJ_ABOVE_BGND;
               ISGETP(fm,r,x1+NUM_ADJ_ABOVE_BGND,n,pix);
               j = 0;
               while (x2==0 && j<n-NUM_ADJ_ABOVE_BGND) {
                    z=0;
                    j++;
                    while (pix[j]<bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                         z++;
                         j++;
                    }
                    if (z==NUM_ADJ_ABOVE_BGND)
                         x2=j-NUM_ADJ_ABOVE_BGND+x1;
               }
          }
     }
}
if (x2==0) {
     printf(" No Image Detected %3d ",x1);
     bgrnd=max((bgrnd-1),MIN_BGRND);
     goto errexit;
}
dx = x2-x1;
```

```
     sx=(dx>>1)+x1;

/*** found object in x, now do y ***/

     row=454;
     while (y1==0 && row>2) {
          row -= 2;

          z=0;
          ISGETP(fm,row,sx,1,vpix);
          while (vpix[0]>bgrnd && z<10) {
               z++;
               row-=2;
               ISGETP(fm,row,sx,1,vpix);
          }
          if (z==10)
               y1 = row+20;
     }
     while (y2==0 && row>2) {
          row -= 2;
          z=0;
          ISGETP(fm,row,sx,1,vpix);
          if (vpix[0] >= satlevel) {
               sat = 1;
          }
          while (vpix[0] < bgrnd && z < 10) {
               z++;
               row-=2;
               ISGETP(fm,row,sx,1,vpix);
          }
          if (z==10)
               y2 = row + 20;
     }

     if (y2==0) {
          printf(" No Image Detected y1 %3d x1 %3d x2 %3d ",y1,x1,x2);
          bgrnd=max((bgrnd-1),MIN_BGRND);
          goto errexit;
     }
     dy = y1-y2;
     sy=(dy>>1)+y2;

/*** found object in y, now redo x at the y midpt for a better result ***/

     x1 = x2 = j = 0;
     n = 260;
     ISGETP(fm,sy&0xfffe,0,260,pix); /* get a line of the nearest even row */
     while (x1==0 && j<n-NUM_ADJ_ABOVE_BGND) {
          z = 0;
          j++;
          while (pix[j]>bgrnd && z<NUM_ADJ_ABOVE_BGND) {
```

```
                    z++;
                    j++;
               }
               if (z==NUM_ADJ_ABOVE_BGND)
                    x1=j-NUM_ADJ_ABOVE_BGND;
          }
          if (x1) {
               while (x2==0 && j<n-NUM_ADJ_ABOVE_BGND) {
                    z=0;
                    j++;
                    while (pix[j]<bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                         z++;
                         j++;
                    }
                    if (z==NUM_ADJ_ABOVE_BGND)
                         x2=j-NUM_ADJ_ABOVE_BGND;
               }
          }
          else {
               printf(" Oddly shaped image -- cannot locate. y1 %3d x1 %3d x2
%3d",y1,x1,x2);
               goto errexit;
          }
          if (!x2) {
               printf(" Objects blur together -- cannot resolve. y1 %3d x1 %3d x2
%3d",y1,x1,x2);
               bgrnd=min((bgrnd+1),MAX_BGRND);
               goto errexit;
          }
          dx = x2-x1;
          sx=(dx>>1)+x1;

     /*** found image bounds, now pick the appropriate filter ***/

          printf(" Center at %2.2d, %2.2d ",sx,sy);
          centx=sx;
          centy=sy;

          clr = 0;
          while (sat==1 && clr < 6) {
               clr++;
               switch(clr) {
                    case 1:
                         centx = 0.956*sx + 133.59;
                         break;
                    case 2:
                         centx = sx + 256;
                         break;
                    case 3:
                         centx = 0.940*sx + 0.357*sy + 149.28;
                         centy = (-0.327)*sx + 0.940*sy - 98.84;
```

```
                        break;
                    case 4:
                        centx = 0.899*sx + 0.342*sy + 31.53;
                        break;
                    case 5:
                        centx = 0.940*sx + 0.357*sy - 106.72;
                        break;
                }
                col = centx - (dx>>1);
                ISGETP(fm,centy&0xfffe,col,dx,pix);
                sat=0;
                for (i=0;i<dx&&!sat;i++)
                    sat = (pix[i]>=satlevel);

        /*** do histogram of image and pick most common value well above background

                **/
                    for (k=0;k<=255;k++)
                        hist[k]=0;
                    ra = (centy - (dy>>1));
                    if (!stdvid) ra &= 0xfffe; /* data is only on even rows */
                    ca = centx - (dx>>1);
                    for (row=ra;row<=ra+dy;row+=2) {
                        if (ISGETP(fm,row,ca,dx,pix)) printf("%d %d %d %d
%d",sx,sy,dx,dy,row);
                        for (j=0;j<dx;j++) {
                            hist[pix[j]]++;
                        }
                    }
                    for (j=satlevel;j<=255&&!sat;j++) /* check for values above saturation
*/
                        sat=(hist[j]>0);
/*    printf (" x1 %3d x2 %3d y1 %3d y2 %3d dx %3d dy %3d ra %3d ca %3d\n",
       x1,x2,y1,y2,dx,dy,ra,ca);
    for (k=0;k<=255;k++)
            printf(" %3.3d",hist[k]);       */
            }
            if (sat) {
                printf(" 11 images are saturated. ");
                bgrnd=min((bgrnd+1),MAX_BGRND);
                goto errexit;
            }

            histmax = 0;
            hmax = 0;
            for (j=255;!hist[j];j--) ;
            for (i=j;i>=(j+bgrnd)>>1;i--) {
                if (hist[i]>histmax) {
                    histmax = hist[i];
                    hmax = i;
                }
```

```
          }

          printf(" FILTER #       Temp.        A/D ");
          printf(  "   %d          %6.1lf        %4d",
            clr, temp[clr][hmax], hmax);
          control(temp[clr][hmax]); /* send out control voltage based on temperature */
errexit:  if (kbhit()) {
               ans = getch(); /* get keystroke */
               if (ans==0)         /* extended key code */
                    ans = getch()<<8;  /* get 2nd byte */
               if (ans==END)
                    goto finalexit;
          }
          color_time(fm,temp,clr);
          ISOUTS(clr);  /* select false-color palette to match filter */
          ISOTFR(fm); /* display frame */
     }
finalexit:     ISEND();
               fcloseall();
}
```

CONTROL.C

```
#include <stdlib.h>
#include <stdio.h>
#include "\iris\iserrs.h"
#include "\iris\isdefs.h"

void control(temp)
double    temp;

{
}
```

QUICKIE.C

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "\iris\iserrs.h"
#include "\iris\isdefs.h"

#define aaa -5.121e-06
#define bbb 0.4914
#define ccc -2.7572
#define fntemp(y) ((-1*bbb+sqrt(bbb*bbb-4.*aaa*(ccc-y*1000./25.)))/(2.*aaa)+273.)
#define SYNC_EXT 1
#define cls printf("")
#define NUM_ADJ_ABOVE_BGND 5
#define MIN_BGRND   3
#define MAX_BGRND   25
#define HOME (0x4700)
#define END (0x4f00)

double    cslope[6],offset[6];

int  idv(p,m,v)
int  p,m;
unsigned int  *v;

{
double  a,b,c;
int i,j,k;
a=2.;c=3.; /* dummy vars*/

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     if ((inp(0x2ed)&1)!=1)  {
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2)
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ed,(char) 6);
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2)
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(char) p);
     }

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while ((inp(0x2ed)&1)!=1)
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     *v = (inp(0x2ec)&m); /* read the data byte */
```

```
}

void xmv(chan,vlts)
int  chan;
double    *vlts;

{
     /* Put this in when thermocouple measurements are desired. */
}

void xdv(chan,vlts)
int chan;
int vlts[];

{
double  a,b,c;
int i,j,k;
a=2.;c=3.; /* dummy vars*/

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     if ((inp(0x2ed)&1)==1){
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          while ((inp(0x2ed)&1)!=1)
               for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          j = (inp(0x2ec)); /* read the data byte */
     }
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ed,8); /* command code for output-analog */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) chan); /* channel select, 0 or 1, or -1 for both */
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) vlts[0]%256); /* lo byte of voltage */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) vlts[0]/256); /* hi byte of voltage */
     if (chan == -1) { /* output bytes for second channel */
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          while((inp(0x2ed)&2)==2) /* wait until board ready */
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
```

```
            for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
            outp(0x2ec,(char) vlts[1]%256); /* lo byte of voltage */
            for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
            while((inp(0x2ed)&2)==2) /* wait until board ready */
                 for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
            for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
            outp(0x2ec,(char) vlts[1]/256); /* hi byte of voltage */
       }
}

void set_itime(tau)
double    tau;
#define   lv (-10.)
{
double    v,actualv,cpv;
int       analog;
     cpv = 4096/20;
     v = (tau + 19.) / 247.;
     if (v > 5.)     v = 5.;
     if (v < .076)  v = .076;
     analog = (v - lv) * cpv;
     actualv = analog / cpv + lv;
     xdv(0,&analog);
     printf(" Integration time = %lg (ms)",actualv * 253.53 - 16.61);
}

/****************************************************************************/
/*************************** quickie.c main ********************************/
/****************************************************************************/

/* label: finalexit */

main (argc,argv)

int argc;
char *argv[];

{
int  hist[256], tm[8], pix[512], pixa[512], pixb[512], vpix[2], bright[100],
     back[100], bright1[2], bright2[2], back1[2], back2[2], dif1[100],
     dif2[100], fm, value, i, j, k, n, z, r, image, time4, avcnt, bgrnd,
     status, time5, inport, mask, record, gain, chan, x1, y1, x2, y2, dx, dy,
     sx, sy, row, col, clr, centx, centy, nm, ra, ca, histmax, hmax,
     frmav, sat, rs, cs, dif, chr[30], pal[6][3][256], x1a, x1b, rowmin,
     stdvid, satlevel,bs,count,ans;

long      tctemp[3], volts[3],bhist[256];
FILE      *f1;
float     vlts;
double    a,b[6],l[6],c,c2,temp[6][256],calib,eps,tau;
time_t    time1;
```

```
struct    tm   *time2;
char      time3[26], imagestr[30], tempstr[30];

     cls; /* clear computer monitor */
     c2=1.4388;
     value=1;
     image=0;
     status = ISINIT(); /** initialize frame grabber and driver and DT-IRIS **/
     if (status>0)
          goto finalexit;
     ISSYNC(SYNC_EXT); /* set sync to external */
     ISDISP(1); /* turn display on */
     ISOTFR(1); /* select frame buffer one for output (display) */
     ISINFR(0); /* select frame buffer zero for input */
     fm = 0;
     printf(" Enter desired integration time (ms), zero for standard video:        ");
     while (!scanf("%lf",&tau))
          printf("= 0);
     if (stdvid)
          tau = 16.67;
     else
          set_itime(tau); /* select greatest possible integration time less
                              than that requested */

/***** loop acquiring and displaying frames until user is ready to start ***/

     ans=0;
     do {
          if (stdvid) { /* standard video rate */
               ISACQ(fm,1);
          }
          else { /* variable int time */
               ISACQE(fm,1);
               ISWAQC();
 /* this code needed for a DT-2851 frame grabber: */
               /*ISINFR(fm);*/
               /*ISPASS(); *//* pass-thru mode on frame grabber until camera has finished
integrating */
               /*do idv(0,1,&value); while(value==0);*/ /* loop until signal that camera
*/
               /*do idv(0,1,&value); while(value==1); *//* is done integrating */
               /*ISFREZ();*/ /* freeze frame */
          }
          ISOTFR(fm);
          fm = 1 - fm; /* toggle frame betw. 0 and 1 */
          ISINFR(fm);
          printf (" Press HOME key to start; END to exit. ");
          if (kbhit()) {
               ans=getch();
               if (ans==0) /* extended key code */
                    ans=(getch())<<8; /* get 2nd byte and put in upper half of word */
          }
     } while (ans !=HOME && ans!=END);
     if (ans==END) {
          ISEND();
          return;
     }

/*** get dark frames to use for background subtracting ***/

     cls;
     while (kbhit()) getch(); /* clear key buffer */
     printf(" Prepare dark frame.  Press any key when ready, \n or ESC to skip background
subtraction. ");
     if (getch()!=' 10;
          ISDISP(0);
          ISFCLR(0);
          ISFCLR(1);
          ISDTFR(0);

          for (k=0;k<=255;k++) {
               pix[k]=k;
               pix[k+256]=255;
          }
          ISLRLT(0,pix); /* load RLUT -- Resultant Look-up Table */

          for (frmav=0;frmav<avcnt;frmav++) {
               if (stdvid) { /* standard video rate */
                    ISALAQ(25,1,0,1);
               }
               else {  /* variable int time */
                    ISAAQE(25,1,0,1);
                    ISWAQC();
               }
          }
          ISDIVC(1,avcnt,2); /* divide buffer by avcnt to find average backgrnd */
          /*** do histogram of background ***/
          if (stdvid)
               ISHIST(2,bhist);
          else {
               for (k=0;k<=255;k++)
                    bhist[k]=0;
               for (row=2;row<=454;row+=2) {
                    ISGETP(2,row,0,512,pix);
                    for (j=0;j<512;j++) {
                         bhist[pix[j]]++;
                    }
               }
          }
          for (j=0;j<=255;j++)
               if (bhist[j]>50)
```

```
                    satlevel=j; /* pick highest light level significantly present in the
background */
          satlevel=255-satlevel; /* subtract this from 255 to get the saturation light
level */
          bs=1;
     }
     else {
          cls;
          printf ("\n No background subtraction. ");
          IS_SET_CONSTANT(2,0);
          satlevel=255;
          bs=0;
     }
     printf ("Saturation level = %d\n",satlevel);

/*** begin collecting data ***/

     while (kbhit()) getch(); /* clear key buffer */
     printf(" Ready for measurement.  Press any key to start. ");
     getch();
     cls;

     count=0;
     for (k=0;k<=255;k++) {
          pix[k]=0;
          pix[k+256]=k;
     }
     ISLRLT(0,pix); /* load RLUT -- Resultant Look-up Table */
     ISDISP(1);
     while (1) {
          count++;
          fm = 1 - fm;
          printf(" Acquiring... ");
          if (stdvid) { /* standard video rate */
               ISALAQ(38,2,0,fm);
          }
          else { /* variable int time */
               ISAAQE(38,2,0,fm);
               ISWAQC();
          }
          ISOTFR(fm); /* display frame */
          printf(" %6d frames ",count);
          if (kbhit() && (getch()==0) && (getch()<<8==END))
               break;
     }
finalexit: ISEND();
}
```

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "\iris\iserrs.h"
#include "\iris\isdefs.h"

#define aaa -5.121e-06
#define bbb 0.4914
#define ccc -2.7572
#define fntemp(y) ((-1*bbb+sqrt(bbb*bbb-4.*aaa*(ccc-y*1000./25.)))/(2.*aaa)+273.)

#define SYNC_EXT 1

#define cls printf("")

#define NUM_ADJ_ABOVE_BGND 5
#define MIN_BGRND   3
#define MAX_BGRND   25

#define HOME (0x4700)
#define END (0x4f00)
#define TAB 9
#define BACKTAB (0x0F00)
#define RIGHT (0x4d00)
#define LEFT  (0x4b00)
#define UP (0x4800)
#define DOWN (0x5000)
#define CTRL_RIGHT (0x7400)
#define CTRL_LEFT  (0x7300)
#define PAGE_UP (0x4900)
#define PAGE_DOWN (0x5100)
#define EIGHT_KEY (0x38)
#define TWO_KEY (0x32)
#define FOUR_KEY (0x34)
#define SIX_KEY (0x36)
#define ALT_1 (0x7800)
#define ALT_2 (0x7900)
#define ALT_3 (0x7a00)
#define ALT_4 (0x7b00)
#define ALT_5 (0x7c00)
#define ALT_6 (0x7d00)

double    cslope[6],offset[6];

int  idv(p,m,v)
int  p,m;
unsigned int  *v;
```

```
{
double  a,b,c;
int i,j,k;
a=2.;c=3.; /* dummy vars*/

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     if ((inp(0x2ed)&1)!=1)  {
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2)
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ed,(char) 6);
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2)
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(char) p);
     }

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while ((inp(0x2ed)&1)!=1)
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     *v = (inp(0x2ec)&m); /* read the data byte */
}

void xmv(chan,vlts)
int  chan;
double    *vlts;

{
     /* Put this in when thermocouple measurements are desired. */
}

void xdv(chan,vlts)
int chan;
int vlts[];

{
double  a,b,c;
int i,j,k;
a=2.;c=3.; /* dummy vars*/

     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     if ((inp(0x2ed)&1)==1){
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          while ((inp(0x2ed)&1)!=1)
               for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          j = (inp(0x2ec)); /* read the data byte */
     }
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ed,8); /* command code for output-analog */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) chan); /* channel select, 0 or 1, or -1 for both */
     for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) vlts[0]%256); /* lo byte of voltage */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     while((inp(0x2ed)&2)==2) /* wait until board ready *
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
     outp(0x2ec,(char) vlts[0]/256); /* hi byte of voltage */
     if (chan == -1) { /* output bytes for second channel */
          for (k=0;k<5;k++){ b=sin(a*c);a=cos(b*c); } /* kill time */
          while((inp(0x2ed)&2)==2) /* wait until board ready */
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(char) vlts[1]%256); /* lo byte of voltage */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          while((inp(0x2ed)&2)==2) /* wait until board ready */
               for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          for (k=0;k<5;k++) { b=sin(a*c);a=cos(b*c);} /* kill time */
          outp(0x2ec,(cnar) vlts[1]/256); /* hi byte of voltage */
     }
}

void set_itime(tau) /*** set voltage corresponding to integration time **/
double    tau;
#define   lv (-10.)
{
double    v,actualv,cpv;
int       analog;
     cpv = 4096/20;
     v = (tau + 19.) / 247.;
     if (v > 5.)    v = 5.;
     if (v < .076)  v = .076;
     analog = (v - lv) * cpv;
     actualv = analog / cpv + lv;
     xdv(0,&analog);
     printf(" Integration time = %lg (ms)",actualv * 253.53 - 16.61);
}

void load_palette(c2,l,temp,eps,tau) /*** create palettes and temperatures table ***/
```

```
double    c2,l[6],temp[6][256],eps,tau;

{
FILE      *pf,*tf;
double    t,vo,factor;
int       pl,index,ado,adi,red[2][256],green[2][256],blue[2][256],flag ;

     flag = 0;
     red[0][0]=green[0][0]=blue[0][0]=0;
     red[1][0]=green[1][0]=blue[1][0]=0;
     tf = fopen("temp.dat","w");
     temp[0][0] = 0.;
     pf = fopen("pal1.lut","r");
     for (index=1;index<=255;index++) {
          fscanf(pf,"%d%d%d",&(red[0][index]),&(green[0][index]),
           &(blue[0][index]));
          factor = cslope[0]*eps*tau/
            (50.*l[0]*l[0]*l[0]*l[0]*l[0]*((double)index-offset[0]))+1;
          if (factor<=0)
               flag = index;
          else
               temp[0][index] = c2 / (l[0]*log(factor));
          fprintf(tf,"%3d    %6.1f\n",index,temp[0][index]);
     }
     fclose(pf);
     for (index=0;index<flag;index++) /* in case of positive calibration offset */
          temp[0][index] = temp[0][flag+1];
     ISLDOT(0,&(red[0][0]),&(green[0][0]),&(blue[0][0]));
     printf(" palette 0\n");
     for (pl=1;pl<6;pl++) {
          flag = 0;
          adi = 1;
          temp[pl][0] = 0.;
          for (ado=1;ado<=255;ado++) {
               vo = (double)ado - offset[pl];
               factor = cslope[pl]*eps*tau/
                (16.67*l[pl]*l[pl]*l[pl]*l[pl]*l[pl]*vo) + 1;
               if (factor>0)
                    temp[pl][ado] = c2/(l[pl]*log(factor));
               else {
                    flag = ado;
                    red[pl&1][ado]=green[pl&1][ado]=blue[pl&1][ado]=0;/*black*/
                    break;
               }
               for (t=temp[pl][ado]; t>temp[pl-1][adi] && adi < 256; adi++) ;
               if (adi==256) {
                    for (index=ado;index<=255;index++) {
                         red[pl&1][index] = (red[pl&1][index-1] + 3) % 256;
                         green[pl&1][index] = (green[pl&1][index-1] + 32) % 256;

                         blue[pl&1][index] = (blue[pl&1][index-1] + 64) % 256;
```

```
                        temp[pl][index] = c2/(l[pl]*log(cslope[pl]*eps*tau/
                          (16.67*l[pl]*l[pl]*l[pl]*l[pl]*l[pl]
                          *((double)index-offset[pl])) + 1));
                   }
                   goto done_pal;
              }
              red[pl&1][ado] = red[!(pl&1)][adi];
              green[pl&1][ado] = green[!(pl&1)][adi];
              blue[pl&1][ado] = blue[!(pl&1)][adi];
         }
 done_pal:
         for (index=0;index<flag;index++) /* in case of positive calibration offset */
              temp[pl][index] = temp[pl][flag+1];
         ISLDOT(pl,&(red[pl&1][0]),&(green[pl&1][0]),&(blue[pl&1][0]) );
         printf(" palette %1.1d\n",pl);
    }
    fclose(tf);
}

/*********** color_time -- print color bar at bottom of video **************/

void      color_time(fm,temp,pl)
int  fm;
double temp[6][256];
int pl;

{
int       gpx,gpy,j,idx[256],cn,ct,rn,temparr[10];
char tempstr[10];

     gpy=440;
     gpx=400;
     for (j=0;j<=255;j++)
          idx[j]=j;
     cn=128;
     ct=256;
     for (rn=440;rn<=470;rn++)
          ISPUTP(fm,rn,cn,ct,idx);
     sprintf(tempstr,"%6.1lf",temp[pl][1]);
     IS_SET_GRAPHIC_POSITION (430,128);
     for (j=0;temparr[j]=tempstr[j];j++);
     ISTEXT(fm,j-1,temparr);
     sprintf(tempstr,"%6.1lf",temp[pl][255]);
     IS_SET_GRAPHIC_POSITION (430,384);
     for (j=0;temparr[j]=tempstr[j];j++);
     ISTEXT(fm,j-1,temparr);
}

/************** mprint -- print text on the video monitor *****************/

void mprint(fm,x,y,str)
```

```
int  fm,x,y;
char *str;

{
int  chr[21];
int  ct,h;
     ct = strlen(str);
     for (h=0;h<ct;h++)
          chr[h] = str[h]; /* convert to int array (16 bit) from char array (8 bit) */
     ISGPOS(y,x);
     ISTEXT(fm,ct,chr);
}
/*****************************************************************************/
/************************** manual.c main ********************************/
/*****************************************************************************/

/* labels: errexit,finalexit */

main (argc,argv)

int argc;
char *argv[];

{
int  hist[256], tm[8], pix[512], pixa[512], pixb[512], vpix[2], bright[100],
     back[100], bright1[2], bright2[2], back1[2], back2[2], dif1[100],
     dif2[100], fm, value, i, j, k, n, z, r, image, time4, avcnt, bgrnd,
     status, time5, inport, mask, record, gain, chan, x1, y1, x2, y2, dx, dy,
     sx, sy, row, col, clr, centx, centy, nm, ra, ca, histmax, hmax,
     frmav, sat, rs, cs, dif, chr[30], pal[6][3][256], x1a, x1b, rowmin,
     stdvid, ans, satlevel, bs, count, done, working, x, y, found_image;

long      tctemp[3], volts[3],bhist[256];
FILE      *f1;
double    l[6],c2,temp[6][256],calib,eps,tau,x0,y0,xn,yn,cytmp;
time_t    time1;
struct    tm   *time2;
char      time3[26], imagestr[30], tempstr[30];

     centx=centy=256; /* default in case no image found */
     cls; /* clear computer monitor */
     c2=1.4388;
     value=1;
     image=0;
     status = ISINIT(); /** initialize frame grabber and driver and DT-IRIS **/
     if (status>0)
          goto finalexit;
/*** read in calibration constants ***/
     f1 = fopen("calib.dat","r");
     for (z=0; z<6; z++) {
          fscanf(f1,"%lg%lg",&(cslope[z]),&(offset[z]));
          printf ("calib: cslope %lg offset %lg\n",cslope[z],offset[z]);
     }
     fclose(f1);
     l[0]=0.00009314;
     l[1]=0.00007330;
     l[2]=0.00005900;
     l[3]=0.00004863;
     l[4]=0.00004215;
     l[5]=0.00003605;
     ISSYNC(SYNC_EXT); /* set sync to external */
     ISDISP(1); /* turn display on */
     IS_SELECT_OUTPUT_FRAME(1); /* select frame buffer one for output (display) */
     IS_SELECT_INPUT_FRAME(0); /* select frame buffer zero for input */
     fm = 0;
/*     cls;*/
     printf(" Enter desired integration time (ms), zero for standard video:        ");
     while (!scanf("%lf",&tau))
          printf("= 0);
     if (stdvid)
          tau = 16.67;
     else
          set_itime(tau); /* select closest possible integration time to that
                                    requested */
     bgrnd=(stdvid)?12:7; /* Pick a level to consider as background even after dark frame
is subtracted.  This is modified later as nature of image becomes apparent. */

/*** acquire and display frames until user is ready ***/
     ans = 0;
     do {
          if (stdvid) { /* standard video rate */
               ISACQ(fm,1);
          }
          else { /* variable int time */
               ISACQE(fm,1);
               ISWAQC();
 /* this code needed for a DT-2851 frame grabber: */
               /*IS_SELECT_INPUT_FRAME(fm);*/
               /*IS_PASSTHRU(); *//* pass-thru mode on frame grabber until camera has
finished integrating */
               /*do idv(0,1,&value); while(value==0);*/ /* loop until signal that camera
*/
               /*do idv(0,1,&value); while(value==1); *//* is done integrating *
/
               /*IS_FREEZE_FRAME();*/ /* freeze frame */
          }
          /* color_time(fm,temp,clr); /* color bar and time display */
          IS_SELECT_OUTPUT_FRAME(fm);
          fm = 1 - fm; /* toggle frame betw. 0 and 1 */
          IS_SELECT_INPUT_FRAME(fm);
          printf (" Press HOME key to start; END to exit. ");
          if (kbhit()) {
                ans=getch();
                if (ans==0) /* extended key code */
                     ans=getch()<<8; /* get 2nd byte and put in upper half of word */
           }
      } while (ans !=HOME && ans!=END);
      if (ans==END) {
           ISEND();
           return;
      }
/****/
      cls;
      printf(" Enter emissivity:     ");
      scanf("%lf",&eps);

/*** collect and average dark frames for background subtraction ***/

      cls;
      while (kbhit()) getch(); /* clear key buffer */
      printf(" Prepare dark frame.  Press any key when ready,\n or\
 press ESC to skip background subtraction. ");
      if (getch()==' No background subtraction. ");
           IS_FRAME_CLEAR(2);
           satlevel=255;
           bs=0;
      }
      else {
           cls;
           avcnt = 10;
           ISDISP(0);
           ISFCLR(0);
           ISFCLR(1);
           IS_SELECT_OUTPUT_FRAME(0);

           for (k=0;k<=255;k++) {
                pix[k]=k;
                pix[k+256]=255;
           }
           IS_LOAD_RLUT(0,pix); /* load RLUT -- Resultant Look-up Table */

           for (frmav=0;frmav<avcnt;frmav++) {
                if (stdvid) { /* standard video rate */
                     IS_ALU_ACQUIRE(25,1,0,1);
                }
                else {  /* variable int time */
                     IS_ALU_ACQUIRE_EXTERNAL(25,1,0,1);
                     ISWAQC();
                }
           }
           IS_DIVIDE_CONSTANT(1,avcnt,2); /* divide buffer by avcnt to find average
backgrnd */
           /*** do histogram of background ***/
```

```
          if (stdvid)
               IS_HISTOGRAM(2,bhist);
          else {
               for (k=0;k<=255;k++)
                    bhist[k]=0;
               for (row=2;row<=454;row+=2) {
                    IS_GET_PIXEL(2,row,0,512,pix);
                    for (j=0;j<512;j++) {
                         bhist[pix[j]]++;
                    }
               }
          }
          for (j=0;j<=255;j++)
               if (bhist[j]>50)
                    satlevel=j; /* pick highest light level significantly present in the
background */
          satlevel=255-satlevel; /* subtract this from 255 to get the saturation light
level */
          printf ("Saturation level = %d\n",satlevel);
     }

/*** start taking measurements ***/

     while (kbhit()) getch(); /* clear key buffer */
     printf(" Ready for measurement.  Press any key to start. ");
     getch();
     cls;
     printf ("Loading false-color palettes...\n");
     load_palette(c2,l,temp,eps,tau);
     printf ("...done loading palettes.\n");
     cls;
     printf (" ENTER -- Acquire another image.  ARROW KEYS -- Move cursor.  END -- Quit.
\
n ARROW KEYS (NUMLOCK ON) -- Move cursor fast.  HOME -- Center of this image.
\n +/- KEYS -- Center of next/prev image.
 \n TAB/BACKTAB -- Same place in next/prev image.
  ");

     for (k=0;k<=255;k++) {
          pix[k]=0;
          pix[k+256]=k;
     }
     IS_LOAD_RLUT(0,pix); /* load RLUT -- Resultant Look-up Table */


     ISDISP(1);
     done=0;
     while (!done) {
          clr = 0;
          record = 0;
          sat = 0;
```

```
        fm = 1 - fm;
        printf(" Acquiring... ");
        if (stdvid) { /* standard video rate */
             IS_ALU_ACQUIRE(38,2,0,fm);
        }
        else { /* variable int time */
             IS_ALU_ACQUIRE_EXTERNAL(38,2,0,fm);
             ISWAQC();
        }
        printf("");

/********* find image **********/

        found_image=0;
        n = 100+NUM_ADJ_ABOVE_BGND;
        y1=y2=x1=x2=0;
        for (cs=2;x2==0 && cs<=200;cs+=100) {
             for (r=454; !x2 && r>=154; r-=2) {
                  j = 0;
                  IS_GET_PIXEL(fm,r,cs,n,pix);
                  while (x1==0 && j<n-NUM_ADJ_ABOVE_BGND) {
                       z = 0;
                       j++;
                       while (pix[j]>bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                            z++;
                            j++;
                       }
                       if (z==NUM_ADJ_ABOVE_BGND)
                            x1=j-NUM_ADJ_ABOVE_BGND+cs;
                  }
                  if (x1) {
                       n = 256 - x1 + NUM_ADJ_ABOVE_BGND;
                       IS_GET_PIXEL(fm,r,x1+NUM_ADJ_ABOVE_BGND,n,pix);
                       j = 0;
                       while (x2==0 && j<n-NUM_ADJ_ABOVE_BGND) {
                            z=0;
                            j++;
                            while (pix[j]<bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                                 z++;
                                 j++;
                            }
                            if (z==NUM_ADJ_ABOVE_BGND)
                                 x2=j-NUM_ADJ_ABOVE_BGND+x1;
                       }
                  }
             }
        }
        if (x2==0) {
             printf(" No Image Detected %3d",x1);
             bgrnd=max((bgrnd-1),MIN_BGRND);
             goto errexit;
```

```
     }
     dx = x2-x1;
     sx=(dx>>1)+x1;

/*** found object in x, now do y ***/

     row=454;
     while (y1==0 && row>2) {
          row -= 2;

          z=0;
          IS_GET_PIXEL(fm,row,sx,1,vpix);
          while (vpix[0]>bgrnd && z<10) {
               z++;
               row-=2;
               IS_GET_PIXEL(fm,row,sx,1,vpix);
          }
          if (z==10)
               y1 = row+20;
     }
     while (y2==0 && row>2) {
          row -= 2;
          z=0;
          IS_GET_PIXEL(fm,row,sx,1,vpix);
          if (vpix[0] >= satlevel) {
               sat = 1;
          }
          while (vpix[0] < bgrnd && z < 10) {
               z++;
               row-=2;
               IS_GET_PIXEL(fm,row,sx,1,vpix);
          }
          if (z==10)
               y2 = row + 20;
     }

     if (y2==0) {
          printf(" No Image Detected y1 %3d x1 %3d x2 %3d",y1,x1,x2);
          bgrnd=max((bgrnd-1),MIN_BGRND);
          goto errexit;
     }
     dy = y1-y2;
     sy=(dy>>1)+y2;

/*** found object in y, now redo x at the y midpt for a better result ***/

     x1 = x2 = j = 0;
     n = 260;
     IS_GET_PIXEL(fm,sy&0xfffe,0,260,pix); /* get a line of the nearest even row */
     while (x1==0 && j<n-NUM_ADJ_ABOVE_BGND) {
          z = 0;
```

```
                j++;
                while (pix[j]>bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                     z++;
                     j++;
                }
                if (z==NUM_ADJ_ABOVE_BGND)
                     x1=j-NUM_ADJ_ABOVE_BGND;
           }
           if (x1) {
                while (x2==0 && j<n-NUM_ADJ_ABOVE_BGND) {
                     z=0;
                     j++;
                     while (pix[j]<bgrnd && z<NUM_ADJ_ABOVE_BGND) {
                          z++;
                          j++;
                     }
                     if (z==NUM_ADJ_ABOVE_BGND)
                          x2=j-NUM_ADJ_ABOVE_BGND;
                }
           }
           else {
                printf(" Oddly shaped image -- cannot locate. y1 %3d x1 %3d x2
%3d",y1,x1,x2);
                goto errexit;
           }
           if (!x2) {
                printf(" Objects blur together -- cannot resolve. y1 %3d x1 %3d x2
%3d",y1,x1,x2);
                goto errexit;
           }
           dx = x2-x1;
           sx=(dx>>1)+x1;

     /*** found image bounds, now pick the appropriate filter ***/

           printf(" Center at %2.2d, %2.2d ",sx,sy);
           centx=sx;
           centy=sy;

           clr = 0;
           while (sat && clr < 6) {
                clr++;
                switch(clr) {
                     case 1:
                          centx = 0.956*sx + 133.59 + 0.5;
                          break;
                     case 2:
                          centx = sx + 256;
                          cytmp = sy;
                          break;
                     case 3:
                        centx = 0.940*sx + 0.357*sy + 149.28 + 0.5;
                        cytmp = (-0.327)*sx + 0.940*sy - 98.84;
                        break;
                   case 4:
                        centx = 0.899*sx + 0.342*sy + 31.53 + 0.5;
                        break;
                   case 5:
                        centx = 0.940*sx + 0.357*sy - 106.72 + 0.5;
                        break;
              }
              col = centx - (dx>>1);
              IS_GET_PIXEL(fm,centy&0xfffe,col,dx,pix);
              sat=0;
              for (i=0;i<dx&&!sat;i++)
                   sat = (pix[i]>=satlevel);
              if (!sat) {
                   if (!stdvid) {
                        centy = (cytmp + 1.);
                        centy &= 0xfffe; /* make even */
                   }
                   else {
                        centy = cytmp + 0.5;
                   }

              /*** do histogram of image and pick most common value well above background
**/

                   for (k=0;k<=255;k++)
                        hist[k]=0;
                   ra = (centy - (dy>>1));
                   if (!stdvid) ra &= 0xfffe; /* data is only on even rows */
                   ca = centx - (dx>>1);
                   for (row=ra;row<=ra+dy;row+=2) {
                        IS_GET_PIXEL(fm,row,ca,dx,pix);
                        for (j=0;j<dx;j++) {
                             hist[pix[j]]++;
                        }
                   }
                   for (j=satlevel;j<=255&&!sat;j++) /* check for values above saturation
*/
                        sat=(hist[j]>0);
   /* printf (" x1 %3d x2 %3d y1 %3d y2 %3d dx %3d dy %3d ra %3d ca %3d\n",
          x1,x2,y1,y2,dx,dy,ra,ca);
    for (k=0;k<=255;k++)
         printf(" %3.3d",hist[k]);       */
              }
         }
         if (sat) {
              printf("AAll images are saturated. ");
              bgrnd=min((bgrnd+1),MAX_BGRND);
              goto errexit;
          }

          histmax = 0;
          hmax = 0;
          for (j=255;!hist[j];j--) ;
          for (i=j;i>=(j+bgrnd)>>1;i--) {
               if (hist[i]>histmax) {
                    histmax = hist[i];
                    hmax = i;
               }
          }

/** finish up with this frame and continue **/

          found_image = 1;

          printf(" FILTER #       Temp.        A/D ");
          printf(  "   %d          %6.1lf        %3d",
            clr, temp[clr][hmax], hmax);
errexit:  color_time(fm,temp,clr);
          ISOUTS(clr);  /* select false-color palette to match filter */
          IS_SELECT_OUTPUT_FRAME(fm); /* display frame */
          working=1;
          x=centx;
          y=centy;
          IS_SET_CURSOR_POSITION(y,x);
          IS_CURSOR (1); /* turn cursor on */

          /**** loop to allow user to analyze frame ****/

          while (working) {
               IS_GET_PIXEL(fm,y,x,1,pix);
               printf(" Absolute position       Temp.        A/D        Offset from center
");
               printf(  "  Row %3d  Col %3d       %6.1f        %3d         Row%4d  Col%4d",
                    y,x,temp[clr][pix[0]],pix[0],y-centy,x-centx);
               if (pix[0]>=satlevel)
                    printf("(saturated)");
               else
                    printf("           ");
               ans = getch(); /* get keystroke */
               if (ans==0)        /* extended key code */
                    ans = getch()<<8;  /* get 2nd byte */
               switch (ans) {
                    case END:  /* quit program */
                         done=1;
                         working=0;
                         break;
                    case '\r': /* Enter key -- get another frame */
                         working=0;
                         break;
```

```
case RIGHT:  /* move cursor right 1 column */
     x=min(x+1,511);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case LEFT:   /* move cursor left 1 column */
     x=max(x-1,0);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case UP:   /* move cursor up 1 or 2 rows */
     if (stdvid)
          y=max(y-1,0);
     else
          y=max(y-2,0);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case DOWN:  /* move cursor down 1 or 2 rows */
     if (stdvid)
          y=min(y+1,479);
     else
          y=min(y+2,478);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case CTRL_RIGHT: /* move cursor right 16 columns */
case SIX_KEY:
     x=min(x+16,511);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case CTRL_LEFT:  /* move cursor left 16 columns */
case FOUR_KEY:
     x=max(x-16,0);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case PAGE_UP:    /* move cursor up 16 rows */
case EIGHT_KEY:
     y=max(y-16,0);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case PAGE_DOWN:  /* move cursor down 16 rows */
case TWO_KEY:
     y=min(y+16,478);
     IS_SET_CURSOR_POSITION(y,x);
     break;
case HOME: /* move cursor to center of this image */
     if (found_image) {
          x=centx;
          y=centy;
          IS_SET_CURSOR_POSITION(y,x);
     }
     break;
case ALT_1:
     clr = 0;
```

C.2

```
          ISOUTS(0);
          break;
     case ALT_2:
          clr = 1;
          ISOUTS(1);
          break;
     case ALT_3:
          clr = 2;
          ISOUTS(2);
          break;
     case ALT_4:
          clr = 3;
          ISOUTS(3);
          break;
     case ALT_5:
          clr = 4;
          ISOUTS(4);
          break;
     case ALT_6:
          clr = 5;
          ISOUTS(5);
          break;
     case '+':  /* move cursor to center of next image */
     case '-':  /* move cursor to center of previous image */
          if (!found_image)
               break;
          if (ans=='+')
               clr=min(clr+1,5);
          else
               clr=max(clr-1,0);
          switch(clr) {  /* translate centx & centy to appropriate filter
coords */
               case 0:
                    centx=sx;
                    break;
               case 1:
                    centx = 0.956*sx + 133.59 + 0.5;
                    break;
               case 2:
                    centx = sx + 256;
                    cytmp = sy;
                    break;
               case 3:
                    centx = 0.940*sx + 0.357*sy + 149.28 + 0.5;
                    cytmp = (-0.327)*sx + 0.940*sy - 98.84;
                    break;
               case 4:
                    centx = 0.899*sx + 0.342*sy + 31.53 + 0.5;
                    break;
               case 5:
                    centx = 0.940*sx + 0.357*sy - 106.72 + 0.5;
                    break;
                }
                if (!stdvid) {
                     centy = (cytmp + 1.);
                     centy &= 0xfffe; /* make even */
                }
                else {
                     centy = cytmp + 0.5;
                }
                x=centx;
                y=centy;
                IS_SET_CURSOR_POSITION(y,x); /* place cursor */

           /*** do histogram of image and pick most common value well above
background
                     **/
                for (k=0;k<=255;k++)
                     hist[k]=0;
                ra = (centy - (dy>>1));
                if (!stdvid) ra &= 0xfffe; /* data is only on even rows */
                ca = centx - (dx>>1);
                for (row=ra;row<=ra+dy;row+=2) {
                     if (IS_GET_PIXEL(fm,row,ca,dx,pix)) printf("%d %d %d %d
%d",sx,sy,dx,dy,row);
                     for (j=0;j<dx;j++) {
                          hist[pix[j]]++;
                     }
                }

                histmax = 0;
                hmax = 0;
                for (j=255;!hist[j];j--) ;
                for (i=j;i>=(j+bgrnd)>>1;i--) {
                     if (hist[i]>histmax) {
                          histmax = hist[i];
                          hmax = i;
                     }
                }

                printf( "   %d          %6.1f         %3d",
                  clr, temp[clr][hmax], hmax);
                if (hmax>=satlevel)
                     printf("(saturated)");
                else
                     printf("           ");
                break;
           case TAB:    /* move cursor to corresponding place in next image */
           case BACKTAB:/* move cursor to corresponding place in previous image
*/
                if (!found_image)
                     break;
```

```
xn=x;
yn=y;
switch (clr) {  /* translate x & y to filter 0 coords */
     case 0:
          x0=xn;
          y0=yn;
          break;
     case 1:
          x0=(xn-133.59)*1.046025;
          y0=yn;
          break;
     case 2:
          x0=xn-256;
          y0=yn;
          break;
     case 3:
          x0=0.9396815*xn-0.356879*yn-175.54957;
          y0=(yn+0.327*x0+98.84)*1.0638298;
          break;
     case 4:
          x0=0.982345*xn-0.3574064*yn-66.299381;
          y0=(yn+0.327*x0+98.84)*1.0638298;
          break;
     case 5:
          x0=0.9396815*xn-0.356879*yn+65.008882;
          y0=(yn+0.327*x0+98.84)*1.0638298;
          break;
}
if (ans==TAB)
     clr=min(clr+1,5);
else
     clr=max(clr-1,0);
switch(clr) {  /* translate centx, centy, x & y to appropriate
filter coords */
     case 0:
          centx=sx;
          x=x0 + 0.5;
          break;
     case 1:
          centx = 0.956*sx + 133.59 + 0.5;
          x = 0.956*x0 + 133.59 + 0.5;
          break;
     case 2:
          centx = sx + 256;
          cytmp = sy;
          x = x0 + 256 + 0.5;
          yn = y0;
          break;
     case 3:
          centx = 0.940*sx + 0.357*sy + 149.28 + 0.5;
          cytmp = (-0.327)*sx + 0.940*sy - 98.84;
```

```
                x = 0.940*x0 + 0.357*y0 + 149.28 + 0.5;
                yn = (-0.327)*x0 + 0.940*y0 - 98.84;
                break;
            case 4:
                centx = 0.899*sx + 0.342*sy + 31.53 + 0.5;
                x = 0.899*x0 + 0.342*y0 + 31.53 + 0.5;
                break;
            case 5:
                centx = 0.940*sx + 0.357*sy - 106.72 + 0.5;
                x = 0.940*x0 + 0.357*y0 - 106.72 + 0.5;
                break;
        }
        if (!stdvid) {
            centy = (cytmp + 1.); centy &= 0xfffe; /* make even */
            y = (yn + 1.); y &= 0xfffe;
        }
        else {
            centy = cytmp + 0.5;
            y = yn + 0.5;
        }
        IS_SET_CURSOR_POSITION(y,x); /* place cursor */

    /*** do histogram of image and pick most common value well above
background
            **/
        for (k=0;k<=255;k++)
            hist[k]=0;
        ra = (centy - (dy>>1));
        if (!stdvid) ra &= 0xfffe; /* data is only on even rows */
        ca = centx - (dx>>1);
        for (row=ra;row<=ra+dy;row+=2) {
            if (IS_GET_PIXEL(fm,row,ca,dx,pix)) printf("%d %d %d %d
%d",sx,sy,dx,dy,row);
            for (j=0;j<dx;j++) {
                hist[pix[j]]++;
            }
        }

        histmax = 0;
        hmax = 0;
        for (j=255;!hist[j];j--) ;
        for (i=j;i>=(j+bgrnd)>>1;i--) {
            if (hist[i]>histmax) {
                histmax = hist[i];
                hmax = i;
            }
        }

        printf( "   %d          %6.1lf         %3d",
          clr, temp[clr][hmax], hmax);
        if (hmax>=satlevel)
```

```
                                        printf("(saturated)");
                                else
                                        printf("            ");
                                break;
                        default:
                                break;
                }
            }
        }
finalexit:      ISEND();
                fcloseall();
}
```